sci_analysis Documentation

Release 2.2.0

Chris Morrow

Jul 22, 2019

Contents

An easy to use and powerful python-based data exploration and analysis tool

CHAPTER 1

Current Version:

2.2 — Released January 5, 2019

CHAPTER 2

What is sci-analysis?

sci-analysis is a python package for quickly performing exploratory data analysis (EDA). It aims to make performing EDA easier for newcomers and experienced data analysts alike by abstracting away the specific SciPy, NumPy, and Matplotlib commands. This is accomplished by using sci-analysis's `analyze()` function.

The main features of sci-analysis are:

- Fast EDA with the analyze() function.
- Great looking graphs without writing several lines of matplotlib code.
- Automatic use of the most appropriate hypothesis test for the supplied data.
- Automatic handling of missing values.

Currently, sci-analysis is capable of performing four common statistical analysis techniques:

- Histograms and summary of numeric data
- Histograms and frequency of categorical data
- Bivariate and linear regression analysis
- Location testing

CHAPTER 3

What's new in sci-analysis version 2.2?

- Version 2.2 adds the ability to add data labels to scatter plots.
- The default behavior of the histogram and statistics was changed from assuming a sample, to assuming a population.
- Fixed a bug involving the Mann Whitney U test, where the minimum size was set incorrectly.
- Verified compatibility with python 3.7.

CHAPTER 4

Requirements

- Packages: pandas, numpy, scipy, matplotlib, six
- Supports python 2.7, 3.5, 3.6, and 3.7

Bugs can be reported here:

https://github.com/cmmorrow/sci-analysis/issues

CHAPTER 5

Documentation

The documentation on how to install and use sci-analysis can be found here:

http://sci-analysis.readthedocs.io/en/latest/

CHAPTER 6

Table of Contents

6.1 Getting started with sci-analysis

sci-analysis requires python 2.7, 3.5, 3.6, or 3.7.

If one of these four version of python is already installed then this section can be skipped.

If you use MacOS or Linux, python should already be installed. You can check by opening a terminal window and typing `which python` on the command line. To verify what version of python you have installed, type `python --version` at the command line. If the version is 2.7.x, 3.5.x, 3.6.x, or 3.7.x where x is any number, sci-analysis should work properly.

Note: It is not recommended to use sci-analysis with the system installed python. This is because the version of python that comes with your OS will require root permission to manage, might be changed when upgrading the OS, and can break your OS if critical packages are accidentally removed. More info on why the system python should not be used can be found here: https://github.com/MacPython/wiki/wiki/Which-Python

---

If you are on Windows, you might need to install python. You can check to see if python is installed by clicking the Start button, typing `cmd` in the run text box, then type `python.exe` on the command line. If you receive an error message, you need to install python.

The easiest way to install python on any OS is by installing Anaconda or Mini-conda from this page:

https://www.continuum.io/downloads

If you are on MacOS and have GCC installed, python can be installed with homebrew using the command:

```
brew install python
```

If you are on Linux, python can be installed with pyenv using the instructions here: https://github.com/pyenv/pyenv

If you are on Windows, you can download the python binary from the following page, but be warned that compiling the required packages will be required using this method:

https://www.python.org/downloads/windows/

6.1.1 Installing sci-analysis

sci-analysis can be installed with pip by typing the following:

```
pip install sci-analysis
```

On Linux, you can install pip from your OS package manager. If you have Anaconda or Mini-conda, pip should already be installed. Otherwise, you can download pip from the following page:

https://pypi.python.org/pypi/pip

sci-analysis works best in conjunction with the excellent pandas and jupyter notebook python packages. If you don't have either of these packages installed, you can install them by typing the following:

```
pip install pandas
pip install jupyter
```

6.2 Using sci-analysis

From the python interpreter or in the first cell of a Jupyter notebook, type:

```
import numpy as np
import scipy.stats as st
from sci_analysis import analyze
```

This will tell python to import the sci-analysis function `analyze()`.

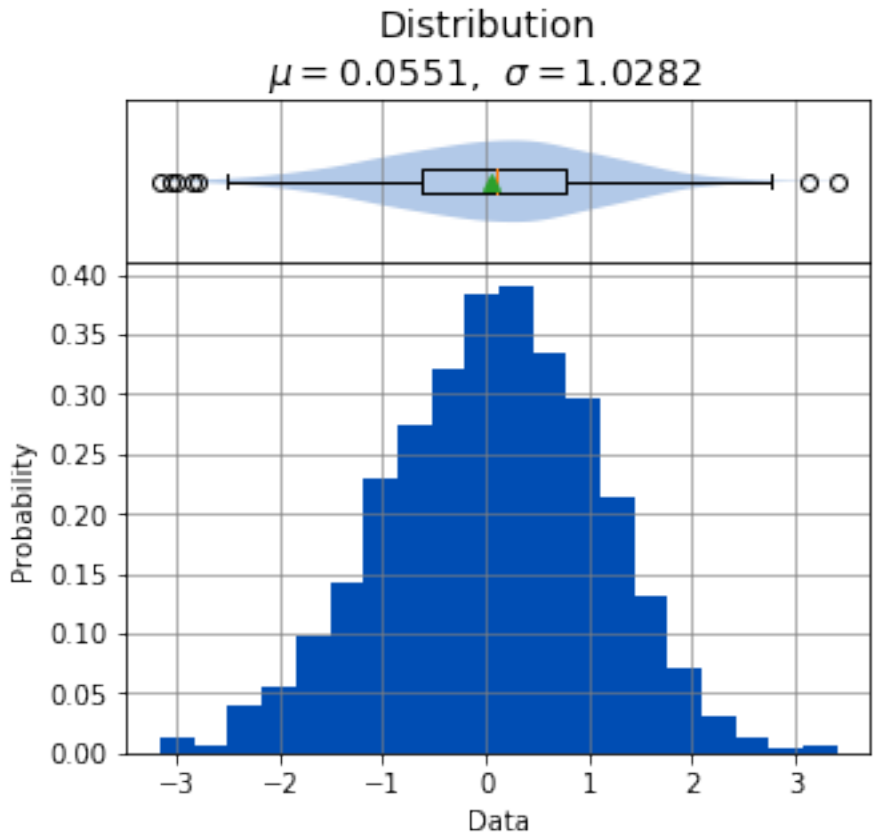
Note: Alternatively, the function `analyse()` can be imported instead, as it is an alias for `analyze()`. For the case of this documentation, `analyze()` will be used for consistency.

If you are using sci-analysis in a Jupyter notebook, you need to use the following code instead to enable inline plots:

```
%matplotlib inline
import numpy as np
import scipy.stats as st
from sci_analysis import analyze
```

Now, sci-analysis should be ready to use. Try the following code:

```
np.random.seed(987654321)
data = st.norm.rvs(size=1000)
analyze(xdata=data)
```



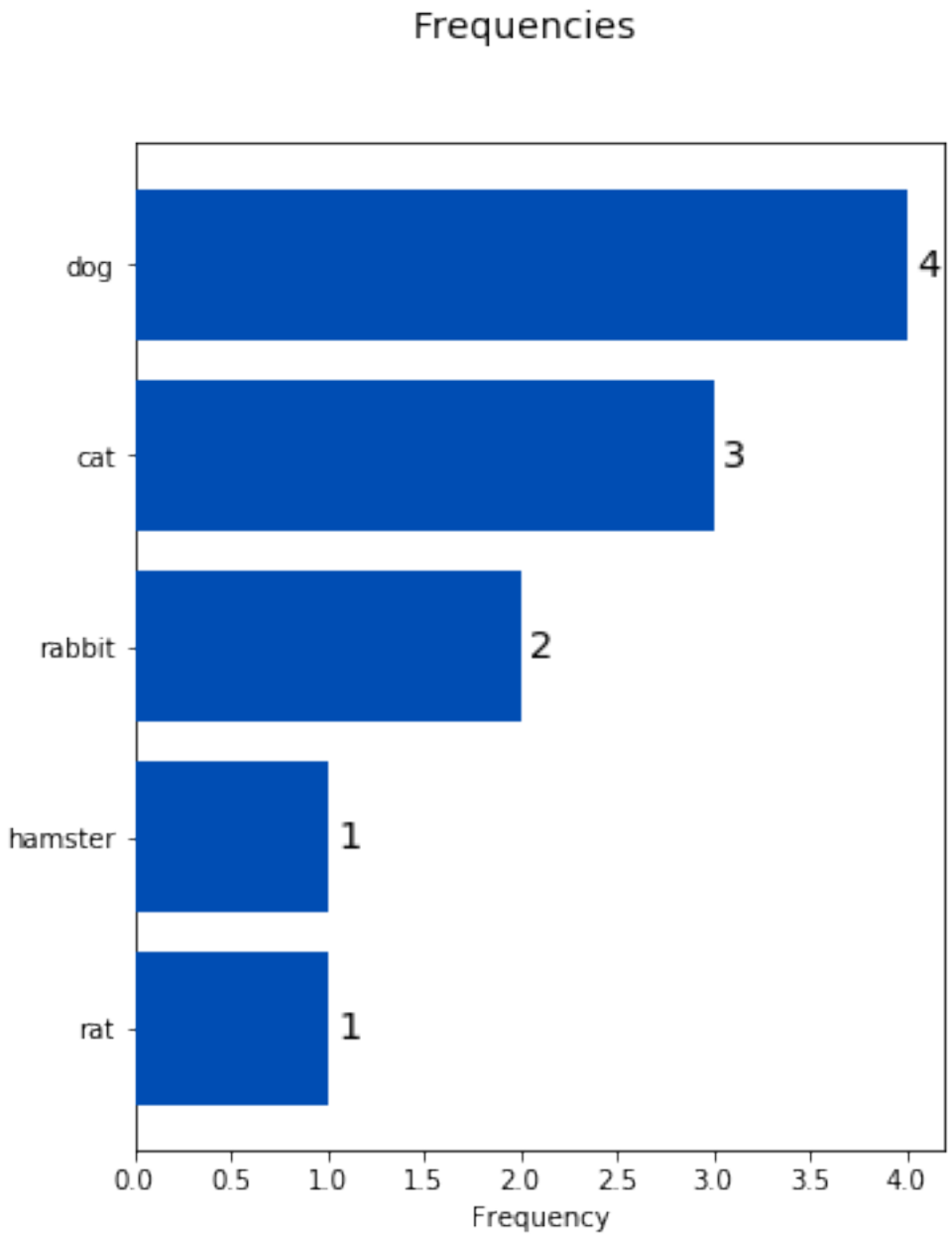

```
Statistics
----------

n          =  1000
Mean       =  0.0551
Std Dev    =  1.0282
Std Error  =  0.0325
Skewness   = -0.1439
Kurtosis   = -0.0931
Maximum    =  3.4087
75%        =  0.7763
50%        =  0.0897
25%        = -0.6324
Minimum    = -3.1586
IQR        =  1.4087
Range      =  6.5673


Shapiro-Wilk test for normality
-------------------------------

alpha    =  0.0500
W value  =  0.9979
p value  =  0.2591

H0: Data is normally distributed
```

A histogram, box plot, summary stats, and test for normality of the data should appear above.

Note: numpy and scipy.stats were only imported for the purpose of the above example. sci-analysis uses numpy and scipy internally, so it isn't necessary to import them unless you want to explicitly use them.

A histogram and statistics for categorical data can be performed with the following command:

```
pets = ['dog', 'cat', 'rat', 'cat', 'rabbit', 'dog', 'hamster', 'cat', 'rabbit', 'dog
↪', 'dog']
analyze(pets)
```




```
Overall Statistics
------------------
```

(continues on next page)

(continued from previous page)

```
Total            =  11
Number of Groups =  5


Statistics
----------

Rank          Frequency     Percent       Category
-------------------------------------------------------
1             4              36.3636      dog
2             3              27.2727      cat
3             2              18.1818      rabbit
4             1              9.0909       hamster
4             1              9.0909       rat
```

Let's examine the `analyze()` function in more detail. Here's the signature for the `analyze()` function:

```
from inspect import signature
print(analyze.__name__, signature(analyze))
print(analyze.__doc__)
```

```
analyze (xdata, ydata=None, groups=None, labels=None, alpha=0.05, order=None,␣
↪dropna=None, **kwargs)

    Automatically performs a statistical analysis based on the input arguments.

    Parameters
    ----------
    xdata : array-like
        The primary set of data.
    ydata : array-like
        The response or secondary set of data.
    groups : array-like
        The group names used for location testing or Bivariate analysis.
    labels : array-like or None
        The sequence of data point labels.
    alpha : float
        The sensitivity to use for hypothesis tests.
    order : array-like
        The order that categories in sequence should appear.
    dropna : bool
        Remove all occurances of numpy NaN.

    Returns
    -------
    xdata, ydata : tuple(array-like, array-like)
        The input xdata and ydata.

    Notes
    -----
    xdata : array-like(num), ydata : None --- Distribution
    xdata : array-like(str), ydata : None --- Frequencies
    xdata : array-like(num), ydata : array-like(num) --- Bivariate
    xdata : array-like(num), ydata : array-like(num), groups : array-like --- Group␣
↪Bivariate
    xdata : list(array-like(num)), ydata : None --- Location Test(unstacked)
```

(continues on next page)

(continued from previous page)

```
    xdata : list(array-like(num)), ydata : None, groups : array-like --- Location␣
↪Test(unstacked)
    xdata : dict(array-like(num)), ydata : None --- Location Test(unstacked)
    xdata : array-like(num), ydata : None, groups : array-like --- Location␣
↪Test(stacked)
```

`analyze()` will detect the desired type of data analysis to perform based on whether the `ydata` argument is supplied, and whether the `xdata` argument is a two-dimensional array-like object.

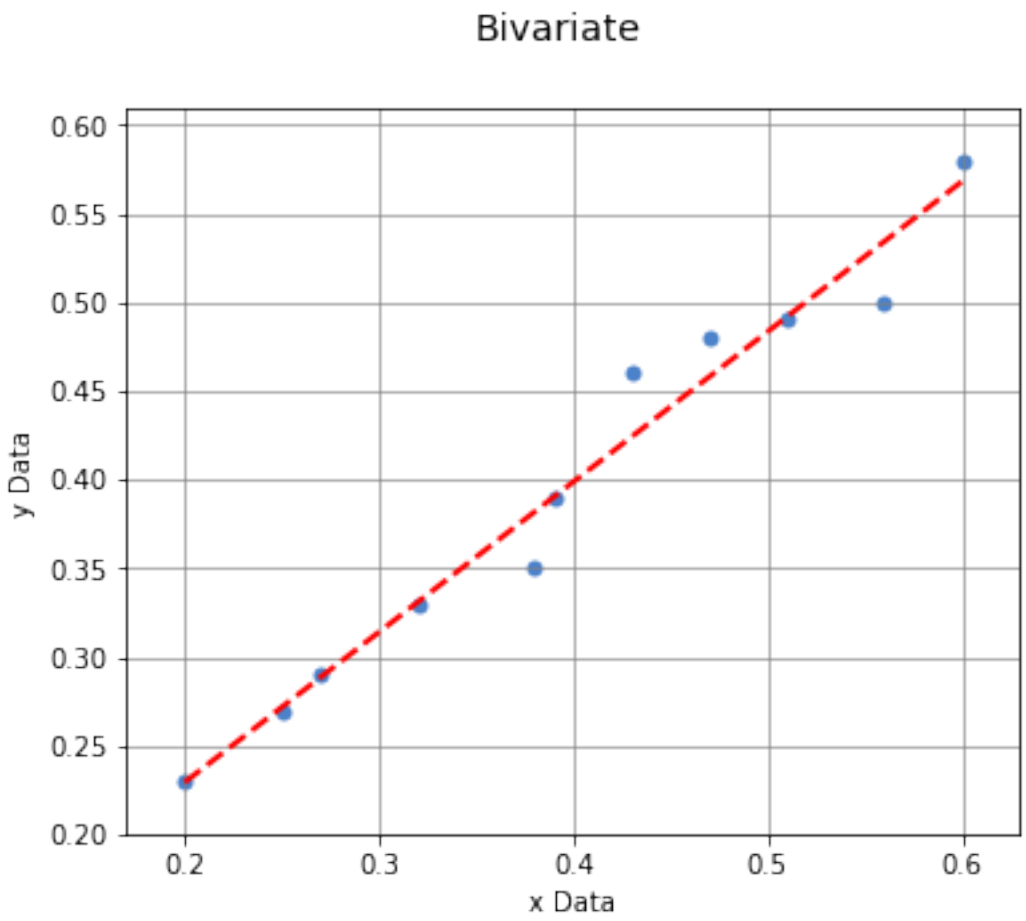
The `xdata` and `ydata` arguments can accept most python array-like objects, with the exception of strings. For example, `xdata` will accept a python list, tuple, numpy array, or a pandas Series object. Internally, iterable objects are converted to a Vector object, which is a pandas Series of type `float64`.

Note: A one-dimensional list, tuple, numpy array, or pandas Series object will all be referred to as a vector throughout the documentation.

If only the `xdata` argument is passed and it is a one-dimensional vector of numeric values, the analysis performed will be a histogram of the vector with basic statistics and Shapiro-Wilk normality test. This is useful for visualizing the distribution of the vector. If only the `xdata` argument is passed and it is a one-dimensional vector of categorical (string) values, the analysis performed will be a histogram of categories with rank, frequencies and percentages displayed.

If `xdata` and `ydata` are supplied and are both equal length one-dimensional vectors of numeric data, an x/y scatter plot with line fit will be graphed and the correlation between the two vectors will be calculated. If there are non-numeric or missing values in either vector, they will be ignored. Only values that are numeric in each vector, at the same index will be included in the correlation. For example, the two following two vectors will yield:

```
example1 = [0.2, 0.25, 0.27, np.nan, 0.32, 0.38, 0.39, np.nan, 0.42, 0.43, 0.47, 0.51,
↪ 0.52, 0.56, 0.6]
example2 = [0.23, 0.27, 0.29, np.nan, 0.33, 0.35, 0.39, 0.42, np.nan, 0.46, 0.48, 0.
↪49, np.nan, 0.5, 0.58]
analyze(example1, example2)
```




```
Linear Regression
-----------------

n         =  11
Slope     =  0.8467
Intercept =  0.0601
r         =  0.9836
r^2       =  0.9674
Std Err   =  0.0518
p value   =  0.0000



Pearson Correlation Coefficient
-------------------------------

alpha   =  0.0500
r value =  0.9836
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

If `xdata` is a sequence or dictionary of vectors, a location test and summary statistics for each vector will be performed. If each vector is normally distributed and they all have equal variance, a one-way ANOVA is performed. If the data is not normally distributed or the vectors do not have equal variance, a non-parametric Kruskal-Wallis test

will be performed instead of a one-way ANOVA.

Note: Vectors should be independent from one another — that is to say, there shouldn't be values in one vector that are derived from or some how related to a value in another vector. These dependencies can lead to weird and often unpredictable results.

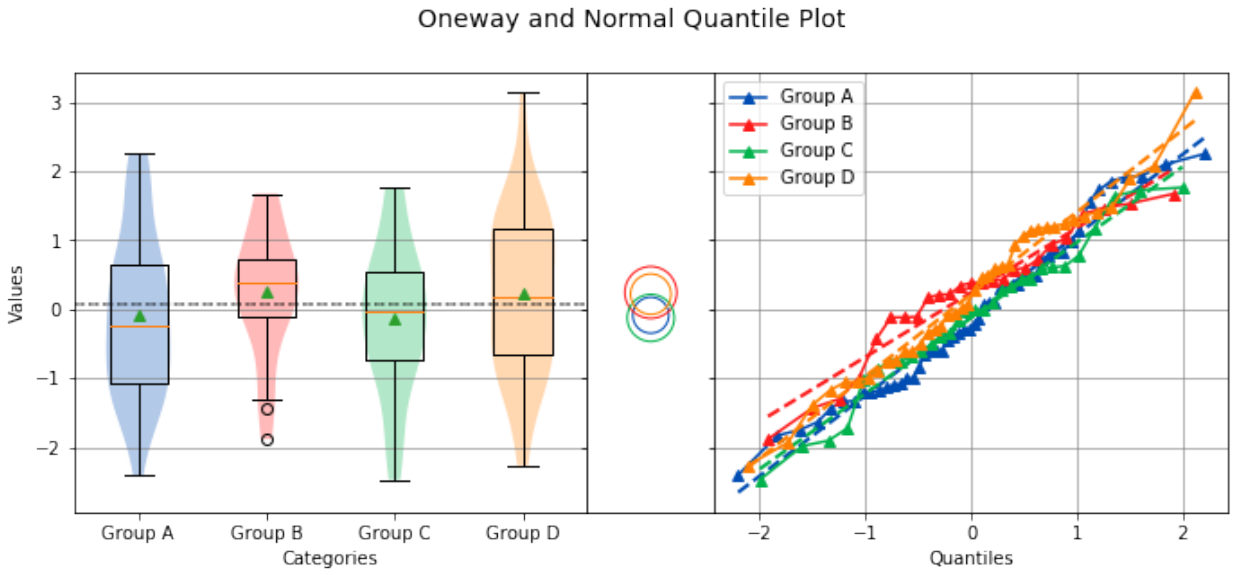
A proper use case for a location test would be if you had a table with measurement data for multiple groups, such as test scores per class, average height per country or measurements per trial run, where the classes, countries, and trials are the groups. In this case, each group should be represented by it's own vector, which are then all wrapped in a dictionary or sequence.

If `xdata` is supplied as a dictionary, the keys are the names of the groups and the values are the array-like objects that represent the vectors. Alternatively, `xdata` can be a python sequence of the vectors and the `groups` argument a list of strings of the group names. The order of the group names should match the order of the vectors passed to `xdata`.

Note: Passing the data for each group into `xdata` as a sequence or dictionary is often referred to as "unstacked" data. With unstacked data, the values for each group are in their own vector. Alternatively, if values are in one vector and group names in another vector of equal length, this format is referred to as "stacked" data. The `analyze()` function can handle either stacked or unstacked data depending on which is most convenient.

For example:

```
np.random.seed(987654321)
group_a = st.norm.rvs(size=50)
group_b = st.norm.rvs(size=25)
group_c = st.norm.rvs(size=30)
group_d = st.norm.rvs(size=40)
analyze({"Group A": group_a, "Group B": group_b, "Group C": group_c, "Group D": group_
↪d})
```




```
Overall Statistics
------------------
```

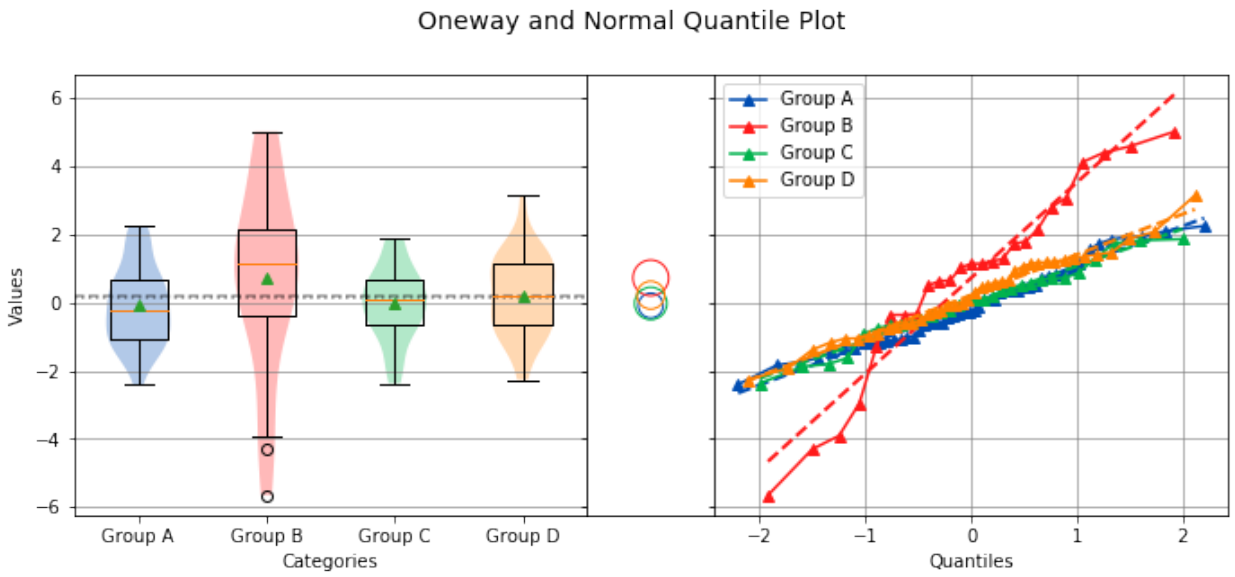
(continues on next page)

(continued from previous page)

```
Number of Groups =  4
Total            =  145
Grand Mean       =  0.0598
Pooled Std Dev   =  1.0992
Grand Median     =  0.0741


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
------------------------------------------------------------------------------------------
↪------------
50            -0.0891        1.1473       -2.4036       -0.2490        2.2466       ␣
↪Group A
25             0.2403        0.9181       -1.8853        0.3791        1.6715       ␣
↪Group B
30            -0.1282        1.0652       -2.4718       -0.0266        1.7617       ␣
↪Group C
40             0.2159        1.1629       -2.2678        0.1747        3.1400       ␣
↪Group D


Bartlett Test
-------------

alpha   =  0.0500
T value =  1.8588
p value =  0.6022

H0: Variances are equal



Oneway ANOVA
------------

alpha   =  0.0500
f value =  1.0813
p value =  0.3591

H0: Group means are matched
```

In the example above, sci-analysis is telling us the four groups are normally distributed (by use of the Bartlett Test, Oneway ANOVA and the near straight line fit on the quantile plot), the groups have equal variance and the groups have matching means. The only significant difference between the four groups is the sample size we specified. Let's try another example, but this time change the variance of group B:

```
np.random.seed(987654321)
group_a = st.norm.rvs(0.0, 1, size=50)
group_b = st.norm.rvs(0.0, 3, size=25)
group_c = st.norm.rvs(0.1, 1, size=30)
group_d = st.norm.rvs(0.0, 1, size=40)
analyze({"Group A": group_a, "Group B": group_b, "Group C": group_c, "Group D": group_
↪d})
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  145
Grand Mean       =  0.2049
Pooled Std Dev   =  1.5350
Grand Median     =  0.1241


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           Group
-------------------------------------------------------------------------------------------
50            -0.0891       1.1473        -2.4036       -0.2490       2.2466        Group A
25             0.7209       2.7543        -5.6558        1.1374       5.0146        Group B
30            -0.0282       1.0652        -2.3718        0.0734       1.8617        Group C
40             0.2159       1.1629        -2.2678        0.1747       3.1400        Group D


Bartlett Test
-------------

alpha   =  0.0500
T value =  42.7597
p value =  0.0000

HA: Variances are not equal
```

(continues on next page)

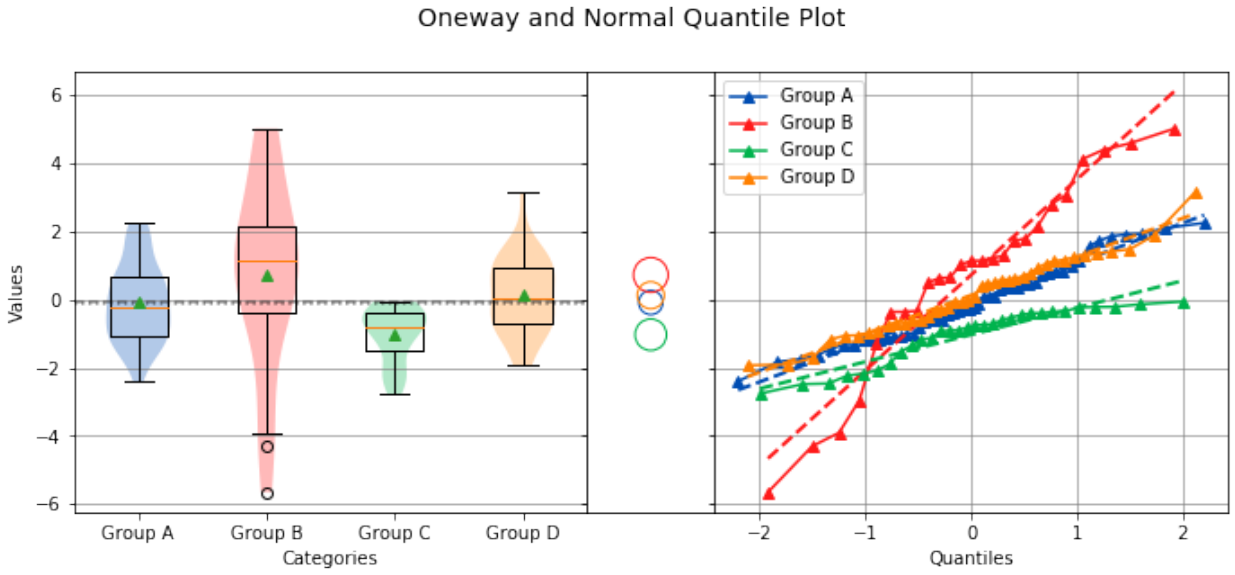
(continued from previous page)

```
Kruskal-Wallis
--------------

alpha    =  0.0500
h value  =  7.1942
p value  =  0.0660

H0: Group means are matched
```

In the example above, group B has a standard deviation of 2.75 compared to the other groups that are approximately 1. The quantile plot on the right also shows group B has a much steeper slope compared to the other groups, implying a larger variance. Also, the Kruskal-Wallis test was used instead of the Oneway ANOVA because the pre-requisite of equal variance was not met.

In another example, let's compare groups that have different distributions and different means:

```
np.random.seed(987654321)
group_a = st.norm.rvs(0.0, 1, size=50)
group_b = st.norm.rvs(0.0, 3, size=25)
group_c = st.weibull_max.rvs(1.2, size=30)
group_d = st.norm.rvs(0.0, 1, size=40)
analyze({"Group A": group_a, "Group B": group_b, "Group C": group_c, "Group D": group_
↪d})
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  145
Grand Mean       = -0.0694
Pooled Std Dev   =  1.4903
Grand Median     = -0.1148


Group Statistics
```

(continues on next page)

(continued from previous page)

```
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
-------------------------------------------------------------------------------------
↪-------------
50            -0.0891        1.1473       -2.4036       -0.2490        2.2466       ␣
↪Group A
25             0.7209        2.7543       -5.6558        1.1374        5.0146       ␣
↪Group B
30            -1.0340        0.8029       -2.7632       -0.7856       -0.0606       ␣
↪Group C
40             0.1246        1.1081       -1.9334        0.0193        3.1400       ␣
↪Group D


Levene Test
-----------

alpha   =  0.0500
W value =  10.1675
p value =  0.0000

HA: Variances are not equal



Kruskal-Wallis
--------------

alpha   =  0.0500
h value =  23.8694
p value =  0.0000

HA: Group means are not matched
```

The above example models group C as a Weibull distribution, while the other groups are normally distributed. You can see the difference in the distributions by the one-sided tail on the group C boxplot, and the curved shape of group C on the quantile plot. Group C also has significantly the lowest mean as indicated by the Tukey-Kramer circles and the Kruskal-Wallis test.

6.3 Using sci-analysis with pandas

Pandas is a python package that simplifies working with tabular or relational data. Because columns and rows of data in a pandas DataFrame are naturally array-like, using pandas with sci-analysis is the preferred way to use sci-analysis.

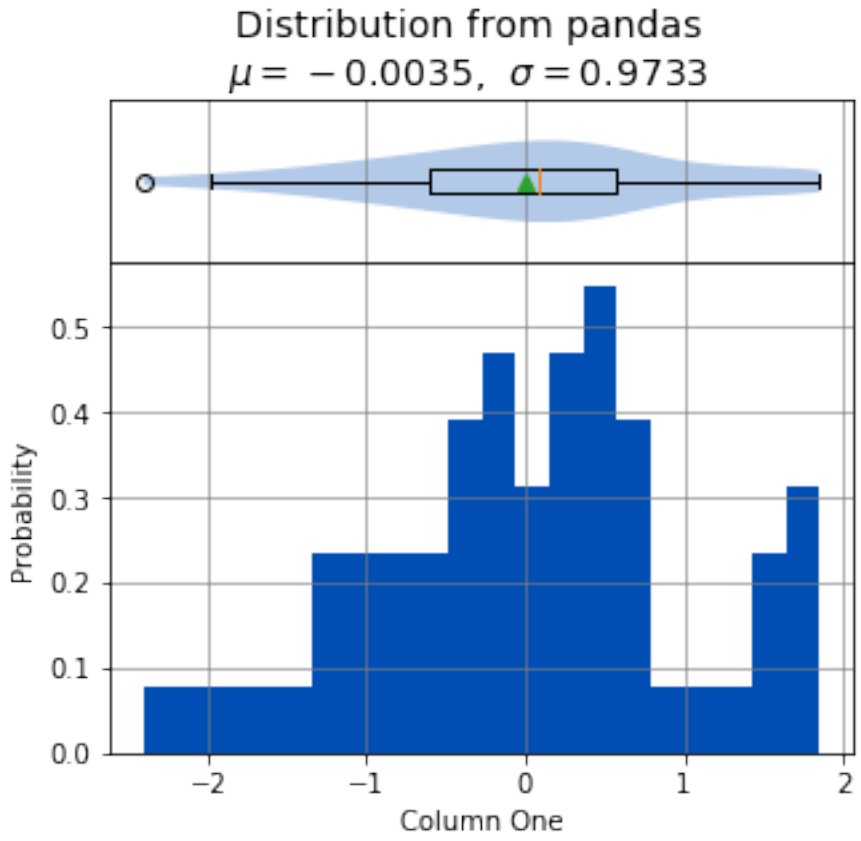
Let's create a pandas DataFrame to use for analysis:

```
%matplotlib inline
import numpy as np
import scipy.stats as st
from sci_analysis import analyze
```

```
import pandas as pd
np.random.seed(987654321)
df = pd.DataFrame(
    {
        'ID'        : np.random.randint(10000, 50000, size=60).astype(str),
        'One'       : st.norm.rvs(0.0, 1, size=60),
        'Two'       : st.norm.rvs(0.0, 3, size=60),
        'Three'     : st.weibull_max.rvs(1.2, size=60),
        'Four'      : st.norm.rvs(0.0, 1, size=60),
        'Month'     : ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
↪'Oct', 'Nov', 'Dec'] * 5,
        'Condition' : ['Group A', 'Group B', 'Group C', 'Group D'] * 15
    }
)
df
```

This creates a table (pandas DataFrame object) with 6 columns and an index which is the row id. The following command can be used to analyze the distribution of the column titled **One**:

```
analyze(
    df['One'],
    name='Column One',
    title='Distribution from pandas'
)
```




```
Statistics
----------
```

(continues on next page)

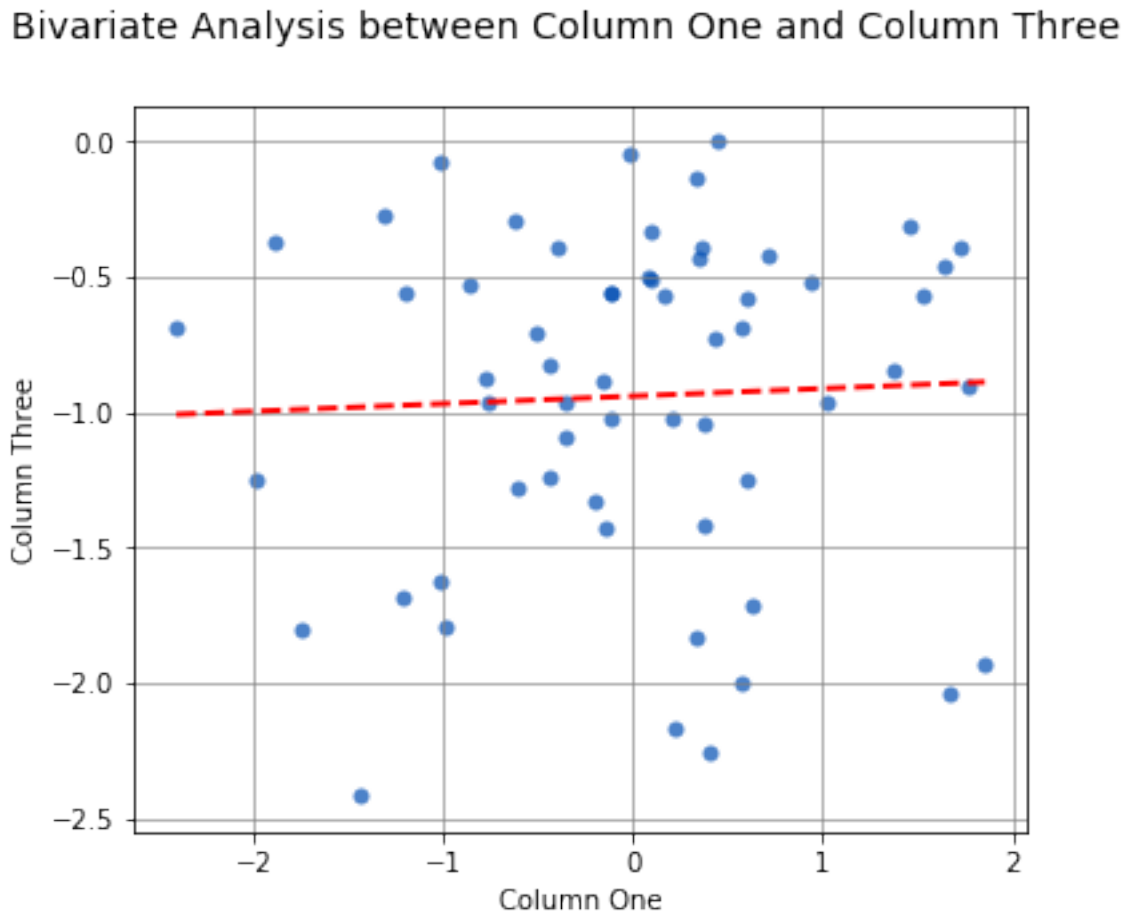
(continued from previous page)

```
n          =  60
Mean       = -0.0035
Std Dev    =  0.9733
Std Error  =  0.1257
Skewness   = -0.1472
Kurtosis   = -0.2412
Maximum    =  1.8537
75%        =  0.5745
50%        =  0.0882
25%        = -0.6113
Minimum    = -2.4036
IQR        =  1.1858
Range      =  4.2573


Shapiro-Wilk test for normality
-------------------------------

alpha   =  0.0500
W value =  0.9804
p value =  0.4460

H0: Data is normally distributed
```

Anywhere you use a python list or numpy Array in sci-analysis, you can use a column or row of a pandas DataFrame (known in pandas terms as a Series). This is because a pandas Series has much of the same behavior as a numpy Array, causing sci-analysis to handle a pandas Series as if it were a numpy Array.

By passing two array-like arguments to the `analyze()` function, the correlation can be determined between the two array-like arguments. The following command can be used to analyze the correlation between columns **One** and **Three**:

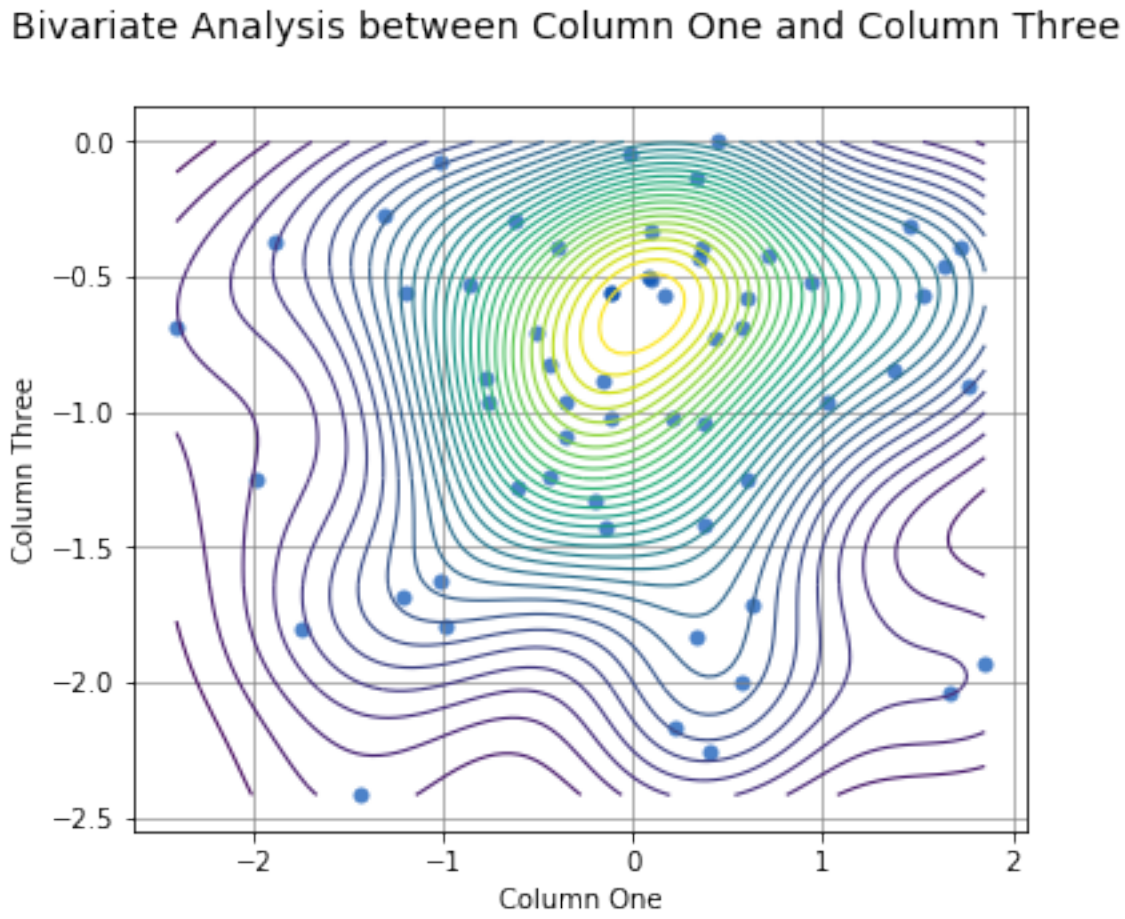
```
analyze(
    df['One'],
    df['Three'],
    xname='Column One',
    yname='Column Three',
    title='Bivariate Analysis between Column One and Column Three'
)
```




```
Linear Regression
-----------------

n          =  60
Slope      =  0.0281
Intercept = -0.9407
r          =  0.0449
r^2        =  0.0020
Std Err    =  0.0820
p value    =  0.7332



Spearman Correlation Coefficient
--------------------------------

alpha   =  0.0500
r value =  0.0316
p value =  0.8105

H0: There is no significant relationship between predictor and response
```

Since there isn't a correlation between columns **One** and **Three**, it might be useful to see where most of the data is concentrated. This can be done by adding the argument `contours=True` and turning off the best fit line with `fit=False`. For example:

```
analyze(
    df['One'],
    df['Three'],
    xname='Column One',
    yname='Column Three',
    contours=True,
    fit=False,
    title='Bivariate Analysis between Column One and Column Three'
)
```




```
Linear Regression
-----------------

n         =  60
Slope     =  0.0281
Intercept = -0.9407
r         =  0.0449
r^2       =  0.0020
Std Err   =  0.0820
p value   =  0.7332



Spearman Correlation Coefficient
```

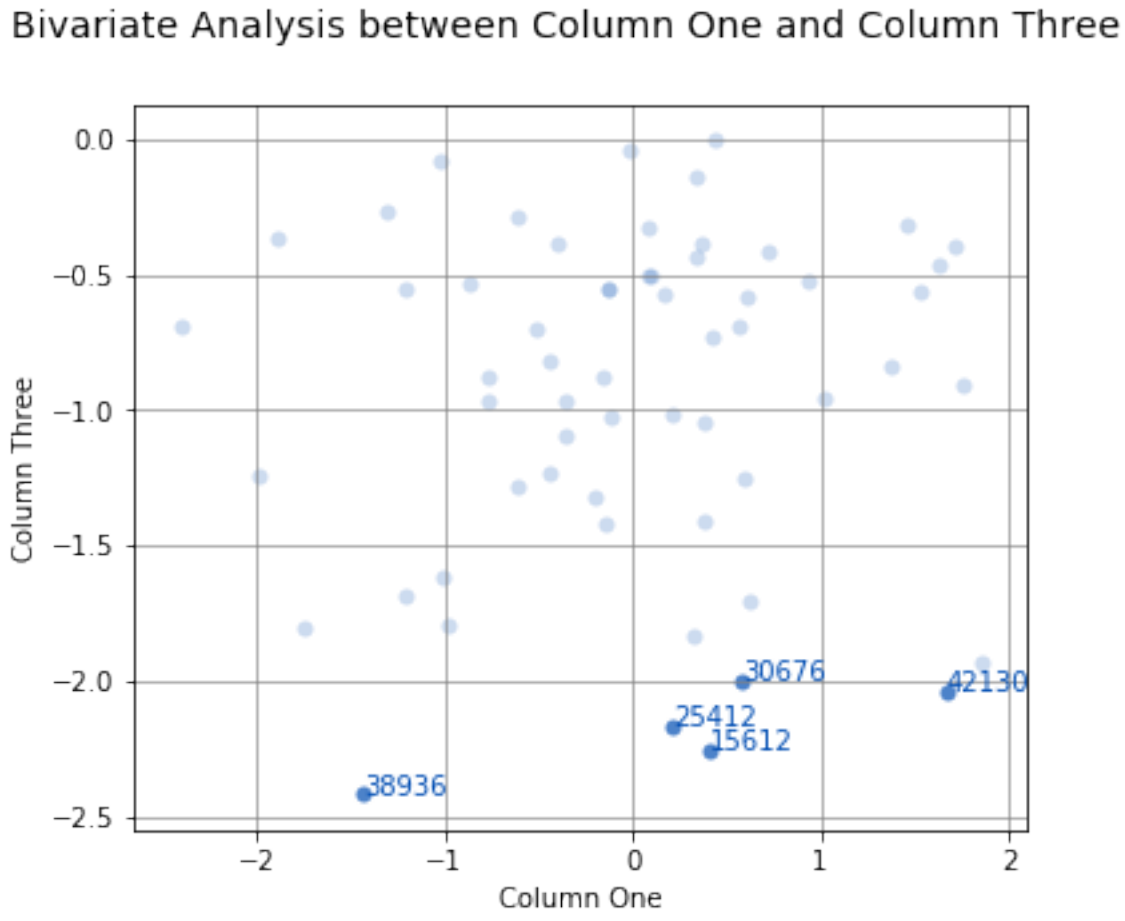
(continues on next page)

(continued from previous page)

```
--------------------------------

alpha   =  0.0500
r value =  0.0316
p value =  0.8105

H0: There is no significant relationship between predictor and response
```

With a few point below -2.0, it might be useful to know which data point they are. This can be done by passing the **ID** column to the `labels` argument and then selecting which labels to highlight with the `highlight` argument:

```
analyze(
    df['One'],
    df['Three'],
    labels=df['ID'],
    highlight=df[df['Three'] < -2.0]['ID'],
    fit=False,
    xname='Column One',
    yname='Column Three',
    title='Bivariate Analysis between Column One and Column Three'
)
```




```
Linear Regression
```

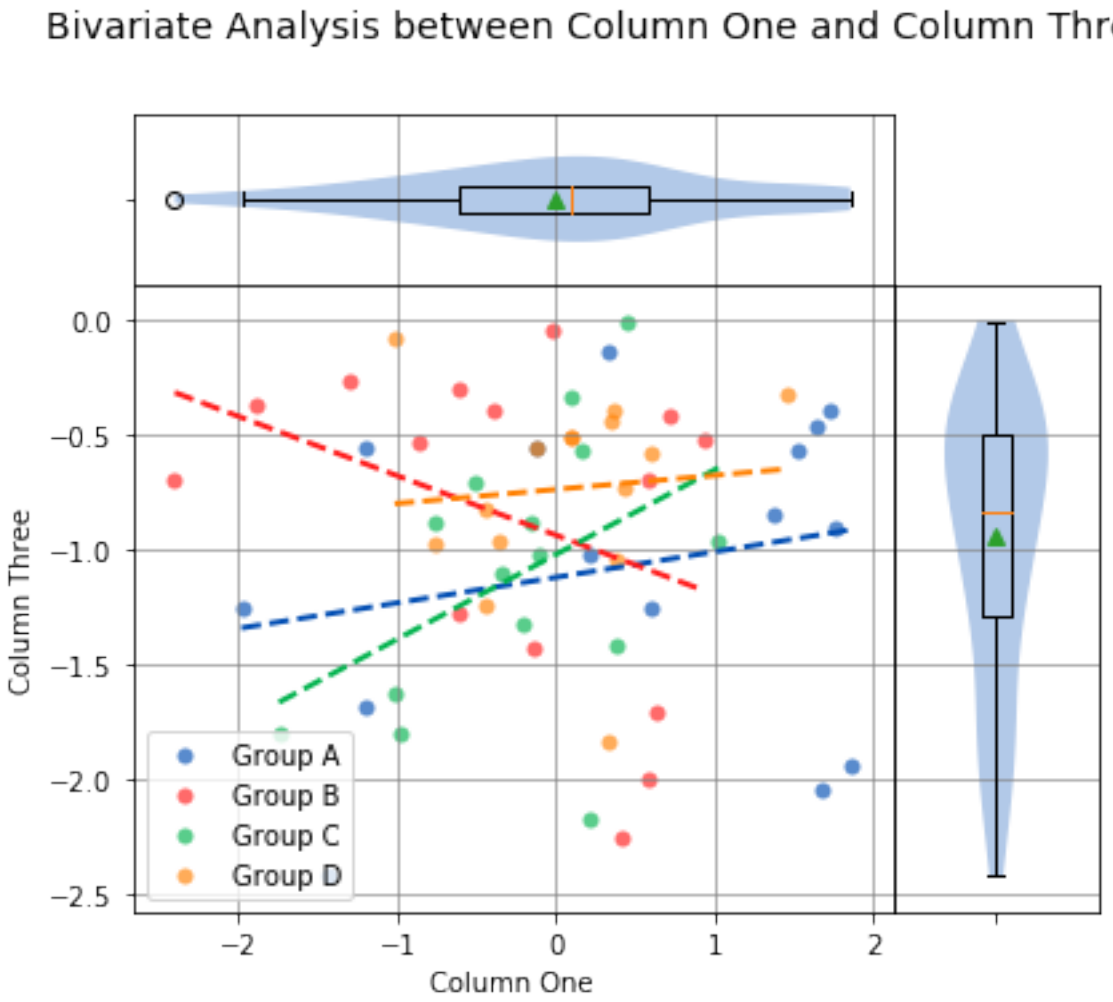
(continues on next page)

(continued from previous page)

```
-----------------

n         =  60
Slope     =  0.0281
Intercept = -0.9407
r         =  0.0449
r^2       =  0.0020
Std Err   =  0.0820
p value   =  0.7332



Spearman Correlation Coefficient
--------------------------------

alpha   =  0.0500
r value =  0.0316
p value =  0.8105

H0: There is no significant relationship between predictor and response
```

To check whether an individual **Condition** correlates between Column **One** and Column **Three**, the same analysis can be done, but this time by passing the **Condition** column to the groups argument. For example:

```
analyze(
    df['One'],
    df['Three'],
    xname='Column One',
    yname='Column Three',
    groups=df['Condition'],
    title='Bivariate Analysis between Column One and Column Three'
)
```




```
Linear Regression
-----------------

n             Slope         Intercept     r^2           Std Err       p value       Group
-------------------------------------------------------------------------------------------
15             0.1113       -1.1181        0.0487        0.1364        0.4293       Group A
15            -0.2586       -0.9348        0.1392        0.1784        0.1708       Group B
15             0.3688       -1.0182        0.1869        0.2134        0.1076       Group C
15             0.0611       -0.7352        0.0075        0.1952        0.7591       Group D


Spearman Correlation Coefficient
--------------------------------

n             r value       p value       Group
-------------------------------------------------------
```

(continues on next page)

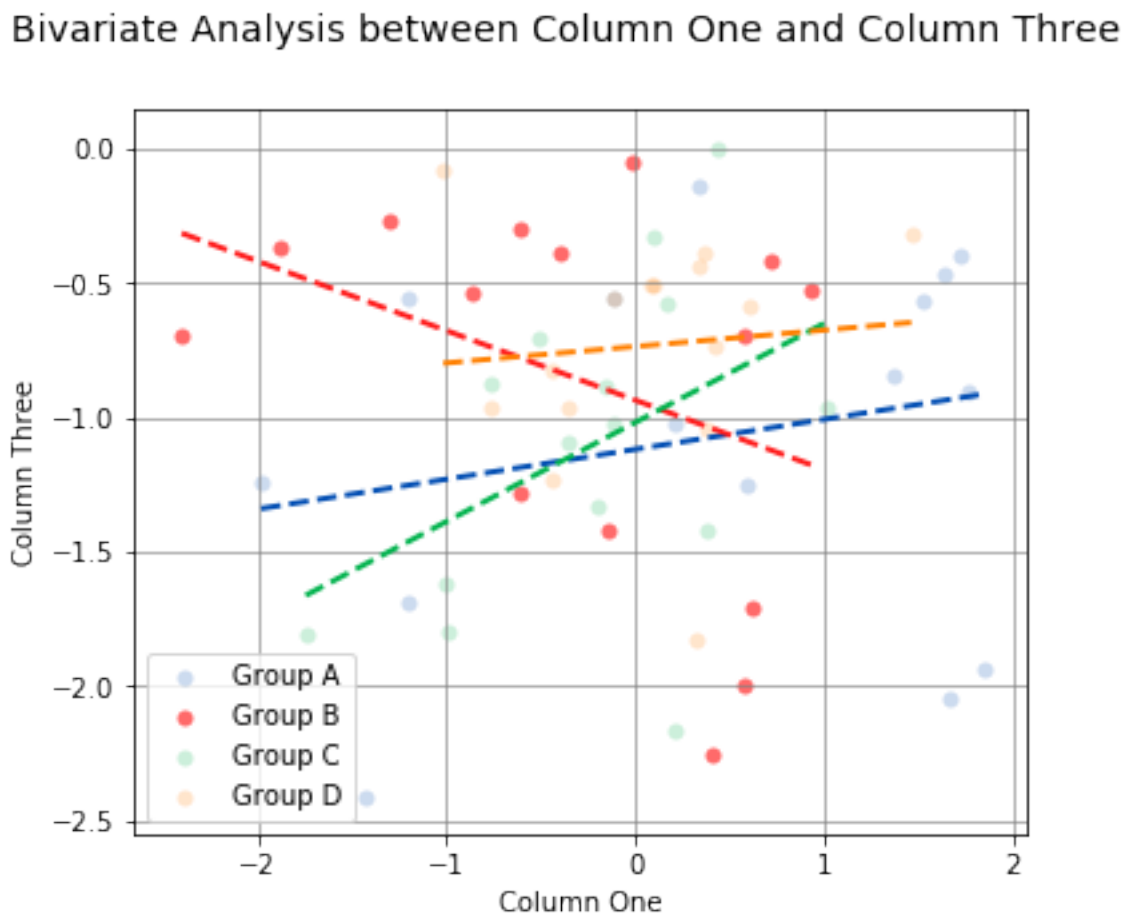
(continued from previous page)

```
15              0.1357        0.6296       Group A
15             -0.3643        0.1819       Group B
15              0.3714        0.1728       Group C
15              0.1786        0.5243       Group D
```

The borders of the graph have boxplots for all the data points on the x-axis and y-axis, regardless of which group they belong to. The borders can be removed by adding the argument `boxplot_borders=False`.

According to the Spearman Correlation, there is no significant correlation among the groups. Group B is the only group with a negative slope, but it can be difficult to see the data points for Group B with so many colors on the graph. The Group B data points can be highlighted by using the argument `highlight=['Group B']`. In fact, any number of groups can be highlighted by passing a list of the group names using the `highlight` argument.

```
analyze(
    df['One'],
    df['Three'],
    xname='Column One',
    yname='Column Three',
    groups=df['Condition'],
    boxplot_borders=False,
    highlight=['Group B'],
    title='Bivariate Analysis between Column One and Column Three'
)
```

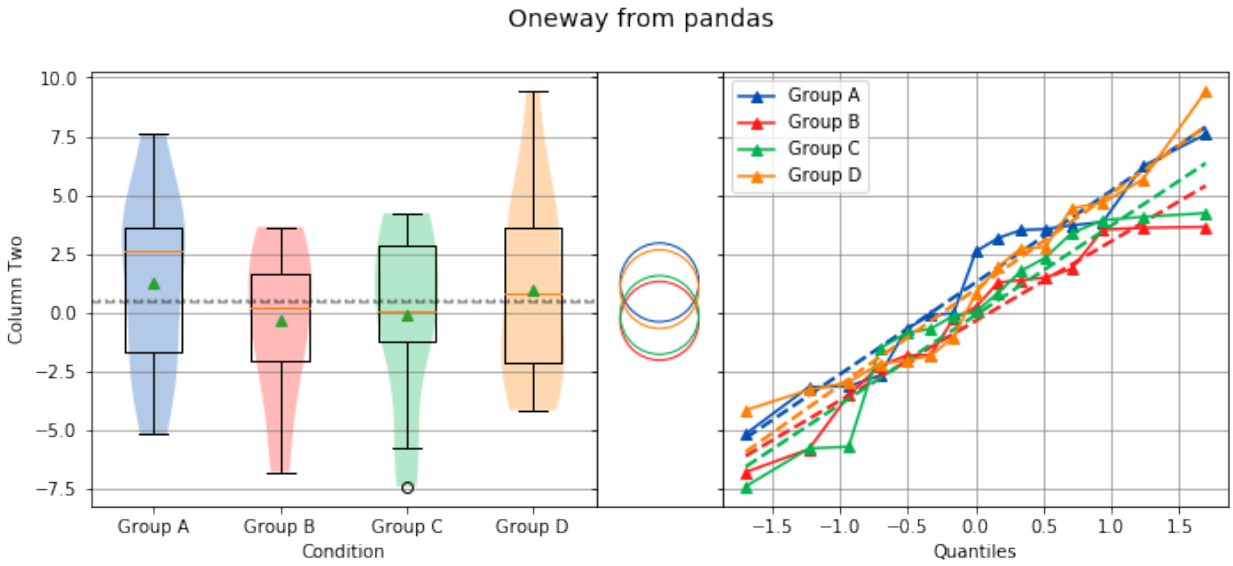


```
Linear Regression
-----------------

n              Slope          Intercept      r^2            Std Err        p value        ⌞
↪Group
-------------------------------------------------------------------------------------------
↪------------
15              0.1113        -1.1181         0.0487         0.1364         0.4293        ⌞
↪Group A
15             -0.2586        -0.9348         0.1392         0.1784         0.1708        ⌞ ⌞
↪Group B
15              0.3688        -1.0182         0.1869         0.2134         0.1076        ⌞ ⌞
↪Group C
15              0.0611        -0.7352         0.0075         0.1952         0.7591        ⌞
↪Group D


Spearman Correlation Coefficient
--------------------------------

n              r value        p value        Group
-----------------------------------------------------
15              0.1357         0.6296        Group A
15             -0.3643         0.1819        Group B
15              0.3714         0.1728        Group C
15              0.1786         0.5243        Group D
```

Performing a location test on data in a pandas DataFrame requires some explanation. A location test can be performed with stacked or unstacked data. One method will be easier than the other depending on how the data to be analyzed is stored. In the example DataFrame used so far, to perform a location test between the groups in the **Condition** column, the stacked method will be easier to use.

Let's start with an example. The following code will perform a location test using each of the four values in the **Condition** column:

```
analyze(
    df['Two'],
    groups=df['Condition'],
    categories='Condition',
    name='Column Two',
    title='Oneway from pandas'
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  60
Grand Mean       =  0.4456
Pooled Std Dev   =  3.6841
Grand Median     =  0.5138


Group Statistics
----------------

n              Mean           Std Dev        Min            Median         Max            Group
-------------------------------------------------------------------------------------------------
15              1.2712         3.7471        -5.1767         2.5874         7.5816        Group A
15             -0.3616         3.2792        -6.8034         0.2217         3.6384        Group B
15             -0.1135         3.7338        -7.4153         0.0224         4.2242        Group C
15              0.9864         3.9441        -4.1688         0.8059         9.4199        Group D


Bartlett Test
-------------

alpha   =  0.0500
T value =  0.4868
p value =  0.9218

H0: Variances are equal
```

(continues on next page)

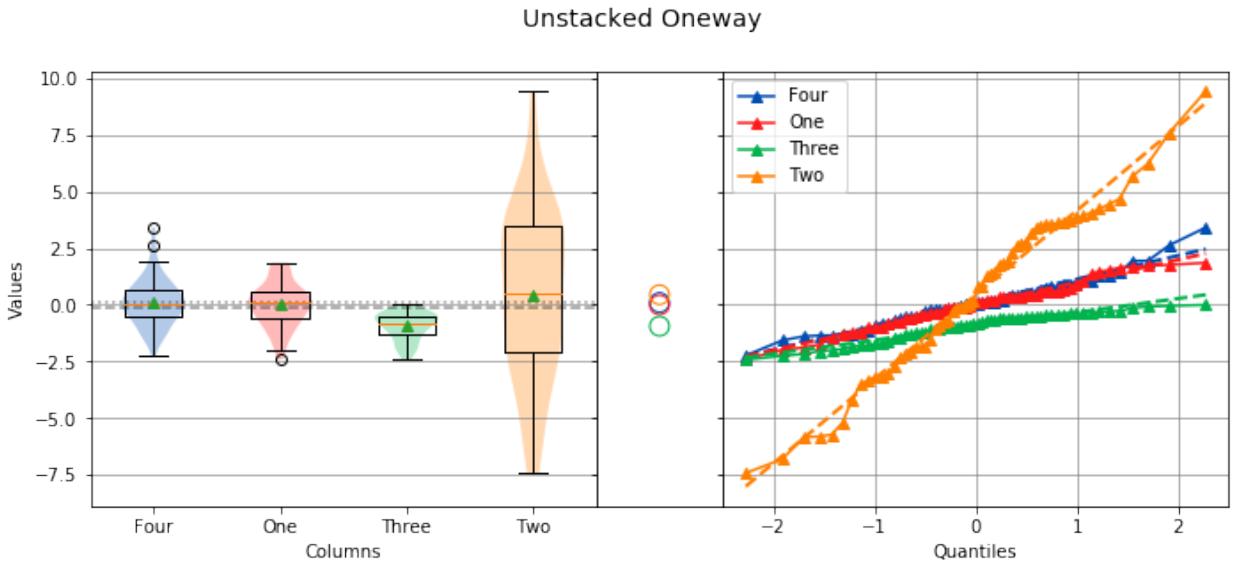
(continued from previous page)

```
Oneway ANOVA
------------

alpha    =  0.0500
f value  =  0.7140
p value  =  0.5477

H0: Group means are matched
```

From the graph, there are four groups: Group A, Group B, Group C and Group D in Column **Two**. The analysis shows that the variances are equal and there is no significant difference in the means. Noting the tests that are being performed, the Bartlett test is being used to check for equal variance because all four groups are normally distributed, and the Oneway ANOVA is being used to test if all means are equal because all four groups are normally distributed and the variances are equal. However, if not all the groups are normally distributed, the Levene Test will be used to check for equal variance instead of the Bartlett Test. Also, if the groups are not normally distributed or the variances are not equal, the Kruskal-Wallis test will be used instead of the Oneway ANOVA.

If instead the four columns **One**, **Two**, **Three** and **Four** are to be analyzed, the easier way to perform the analysis is with the unstacked method. The following code will perform a location test of the four columns:

```
analyze(
    [df['One'], df['Two'], df['Three'], df['Four']],
    groups=['One', 'Two', 'Three', 'Four'],
    categories='Columns',
    title='Unstacked Oneway'
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  240
Grand Mean       = -0.0995
Pooled Std Dev   =  1.9859
Grand Median     =  0.0752
```

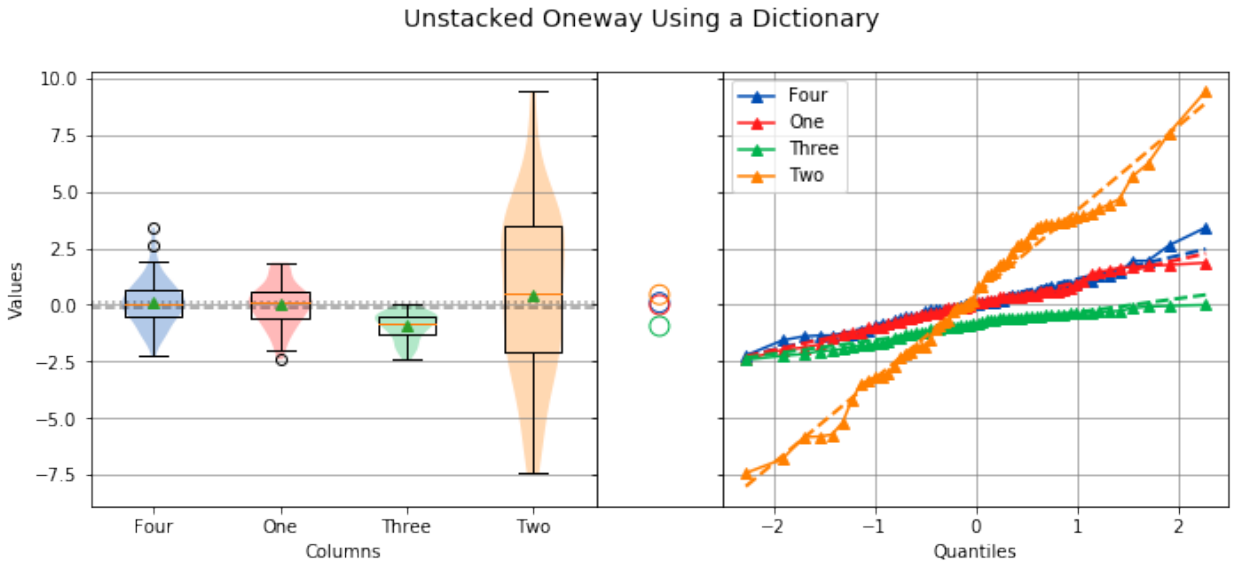
(continues on next page)

(continued from previous page)

```
Group Statistics
----------------

n              Mean           Std Dev        Min            Median         Max            ␣
↪Group
-------------------------------------------------------------------------------------------
↪------------
60              0.1007         1.0294        -2.2349         0.0621         3.4087        ␣
↪Four
60             -0.0035         0.9815        -2.4036         0.0882         1.8537        ␣
↪One
60             -0.9408         0.6133        -2.4177        -0.8318        -0.0015        ␣
↪Three
60              0.4456         3.6572        -7.4153         0.5138         9.4199        ␣
↪Two


Levene Test
-----------

alpha   =  0.0500
W value =  64.7684
p value =  0.0000

HA: Variances are not equal



Kruskal-Wallis
--------------

alpha   =  0.0500
h value =  33.8441
p value =  0.0000

HA: Group means are not matched
```

To perform a location test using the unstacked method, the columns to be analyzed are passed in a list or tuple, and the groups argument needs to be a list or tuple of the group names. One thing to note is that the groups argument was used to explicitly define the group names. This will only work if the group names and order are known in advance. If they are unknown, a dictionary comprehension can be used instead of a list comprehension to to get the group names along with the data:

```
analyze(
    {'One': df['One'], 'Two': df['Two'], 'Three': df['Three'], 'Four': df['Four']},
    categories='Columns',
    title='Unstacked Oneway Using a Dictionary'
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  240
Grand Mean       = -0.0995
Pooled Std Dev   =  1.9859
Grand Median     =  0.0752


Group Statistics
----------------

n              Mean           Std Dev        Min            Median         Max            Group
-------------------------------------------------------------------------------------------------
60              0.1007         1.0294        -2.2349         0.0621         3.4087        Four
60             -0.0035         0.9815        -2.4036         0.0882         1.8537        One
60             -0.9408         0.6133        -2.4177        -0.8318        -0.0015        Three
60              0.4456         3.6572        -7.4153         0.5138         9.4199        Two


Levene Test
-----------

alpha    =  0.0500
W value  =  64.7684
p value  =  0.0000

HA: Variances are not equal
```

(continues on next page)

(continued from previous page)

```
Kruskal-Wallis
--------------

alpha   =  0.0500
h value =  33.8441
p value =  0.0000

HA: Group means are not matched
```

The output will be identical to the previous example. The analysis also shows that the variances are not equal, and the means are not matched. Also, because the data in column **Three** is not normally distributed, the Levene Test is used to test for equal variance instead of the Bartlett Test, and the Kruskal-Wallis Test is used instead of the Oneway ANOVA.

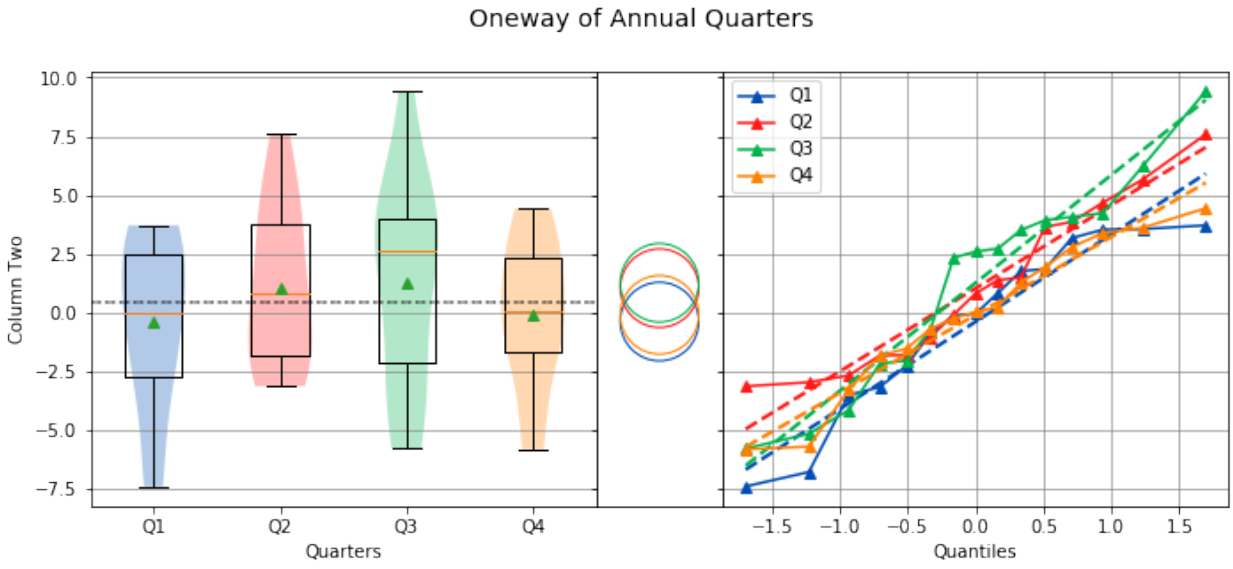
With pandas, it's possible to perform advanced aggregation and filtering functions using the GroupBy object's `apply()` method. Since the sample sizes were small for each month in the above examples, it might be helpful to group the data by annual quarters instead. First, let's create a function that adds a column called **Quarter** to the DataFrame where the value is either Q1, Q2, Q3 or Q4 depending on the month.

```
def set_quarter(data):
    month = data['Month']
    if month.all() in ('Jan', 'Feb', 'Mar'):
        quarter = 'Q1'
    elif month.all() in ('Apr', 'May', 'Jun'):
        quarter = 'Q2'
    elif month.all() in ('Jul', 'Aug', 'Sep'):
        quarter = 'Q3'
    elif month.all() in ('Oct', 'Nov', 'Dec'):
        quarter = 'Q4'
    else:
        quarter = 'Unknown'
    data.loc[:, 'Quarter'] = quarter
    return data
```

This function will take a GroupBy object called *data*, where *data*'s DataFrame object was grouped by month, and set the variable *quarter* based off the month. Then, a new column called **Quarter** is added to *data* where the value of each row is equal to *quarter*. Finally, the resulting DataFrame object is returned.

Using the new function is simple. The same techniques from previous examples are used, but this time, a new DataFrame object called *df2* is created by first grouping by the **Month** column then calling the `apply()` method which will run the `set_quarter()` function.

```
quarters = ('Q1', 'Q2', 'Q3', 'Q4')
df2 = df.groupby(df['Month']).apply(set_quarter)
data = {quarter: data['Two'] for quarter, data in df2.groupby(df2['Quarter'])}
analyze(
    [data[quarter] for quarter in quarters],
    groups=quarters,
    categories='Quarters',
    name='Column Two',
    title='Oneway of Annual Quarters'
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  60
Grand Mean       =  0.4456
Pooled Std Dev   =  3.6815
Grand Median     =  0.4141


Group Statistics
----------------

n              Mean           Std Dev        Min            Median         Max
↪Group
------------------------------------------------------------------------------------------
↪------------
15             -0.3956        3.6190         -7.4153        -0.0510        3.7046
15              1.0271        3.4028         -3.1509         0.8059        7.5816
15              1.2577        4.4120         -5.8003         2.5874        9.4199
15             -0.1067        3.1736         -5.8342         0.0224        4.4318


Bartlett Test
-------------

alpha    =  0.0500
T value  =  1.7209
p value  =  0.6323

H0: Variances are equal


Oneway ANOVA
------------
```

(continues on next page)

(continued from previous page)

```
alpha   =  0.0500
f value =  0.7416
p value =  0.5318

H0: Group means are matched
```

6.4 Analysis Types

6.4.1 Distribution

A distribution describes the spread and tendency of a collection of numeric data. In this case, the spread is the relative distance of a data point to the other data points. You can think of this as data points being grouped close to one another, or spread far apart from one another. A common measurement of this spread is variance, which is the spread from the mean of a distribution.

Mean measures the central tendency of a distribution. Tendency refers to when data points "tend" to group closely to one another. This is easily calculated by summing all the values of the data points and dividing by the total number of data points **n**.

A distribution is represented on a graph by a histogram. A histogram charts a distribution by separating the numeric data in the distribution into discrete bins along the x-axis. These bins are charted as bars, where the height of each bar represents how many numeric values are in that bin.

A distribution represented by a histogram closely resembles a probability density function for continuous numeric data, or a probability mass function for discrete numeric data.

Interpreting the Graphs

The distribution analysis can show three graphs, the histogram, boxplot, and cumulative distribution plot.

Let's first import sci-analysis and setup some variables to use in these examples.

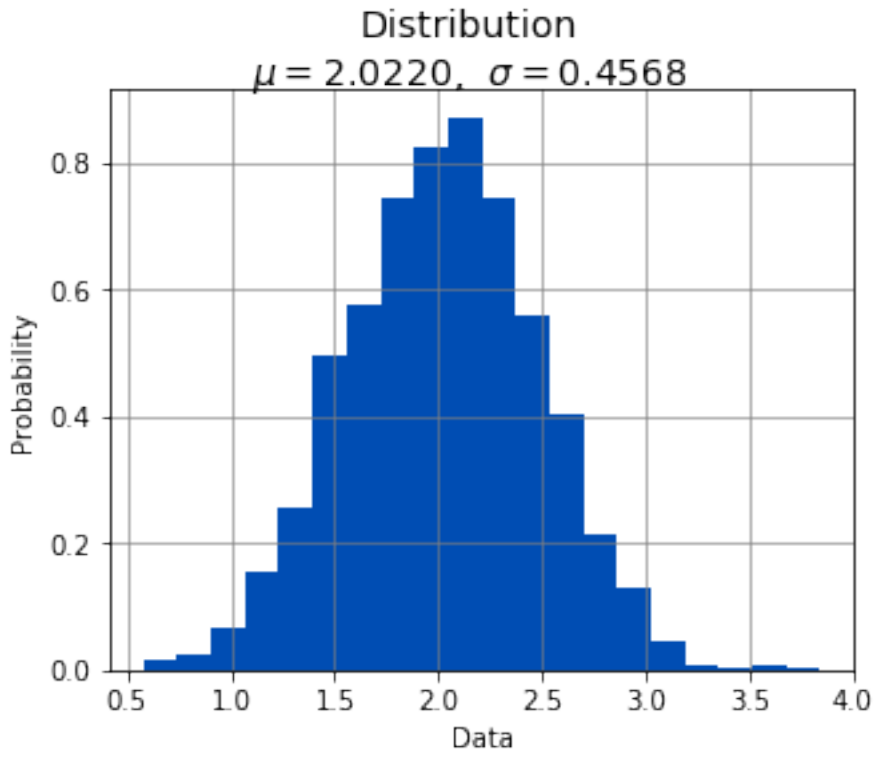
```
import numpy as np
import scipy.stats as st
from sci_analysis import analyze

%matplotlib inline
```

```
# Create sequence from random variables.
np.random.seed(987654321)
sequence = st.norm.rvs(2, 0.45, size=2000)
```

Histogram

The histogram, as described above, separates numeric data into discrete bins along the x-axis. The y-axis is the probability that a data point from a given distribution will belong to a particular bin.

```
analyze(
    sequence,
    boxplot=False,
)
```



Statistics

```
n          =  2000
Mean       =  2.0220
Std Dev    =  0.4568
Std Error  =  0.0102
Skewness   = -0.0017
Kurtosis   = -0.0747
Maximum    =  3.8397
75%        =  2.3453
50%        =  2.0295
25%        =  1.7090
Minimum    =  0.5786
IQR        =  0.6363
Range      =  3.2611
```

Shapiro-Wilk test for normality

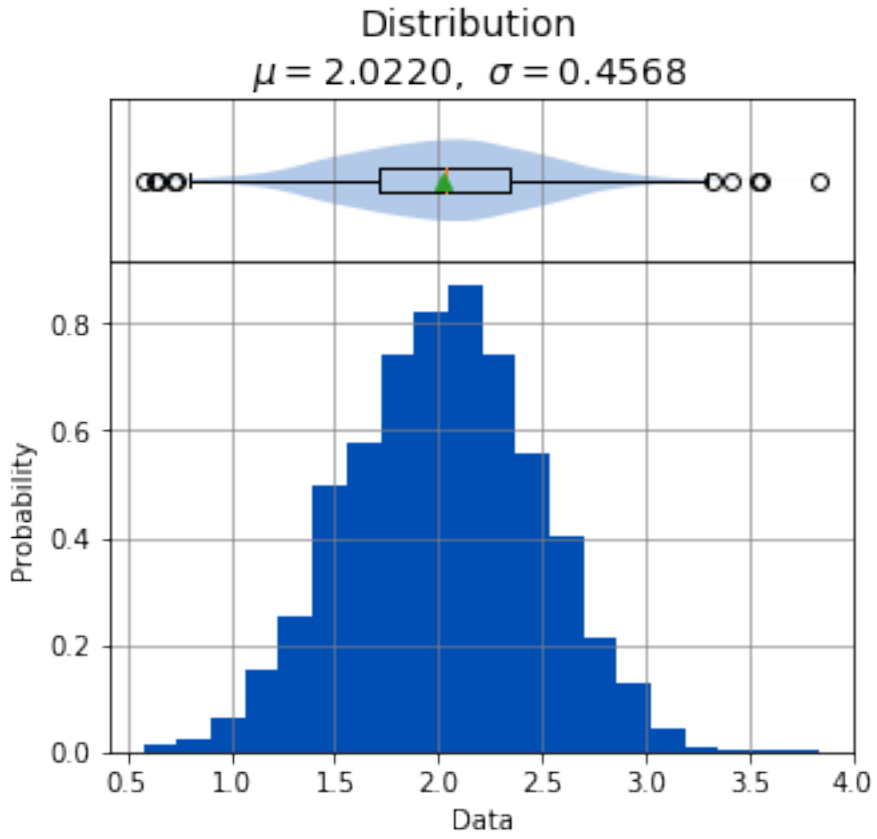
```
alpha    =  0.0500
W value  =  0.9993
p value  =  0.6430

H0: Data is normally distributed
```

Boxplot

Boxplots in sci-analysis are actually a hybrid of two distribution visualization techniques, the boxplot and the violin plot. Boxplots are a good way to quickly understand a distribution, but can be misleading when the distribution is multimodal. A violin plot does a much better job at showing local maxima and minima of a distribution.

In the center of each box is a red line and green triangle. The green triangle represents the mean of the group while the red line represents the median, sometimes referred to as the second quartile (Q2) or 50% line. The circles that might be seen at either end of the boxplot are outliers, and referred to as such because they are in the bottom 5% and top 95% of the distribution.

```
analyze(sequence)
```




```
Statistics
----------

n          =  2000
Mean       =  2.0220
Std Dev    =  0.4568
Std Error  =  0.0102
Skewness   = -0.0017
Kurtosis   = -0.0747
Maximum    =  3.8397
75%        =  2.3453
50%        =  2.0295
25%        =  1.7090
Minimum    =  0.5786
```

(continues on next page)

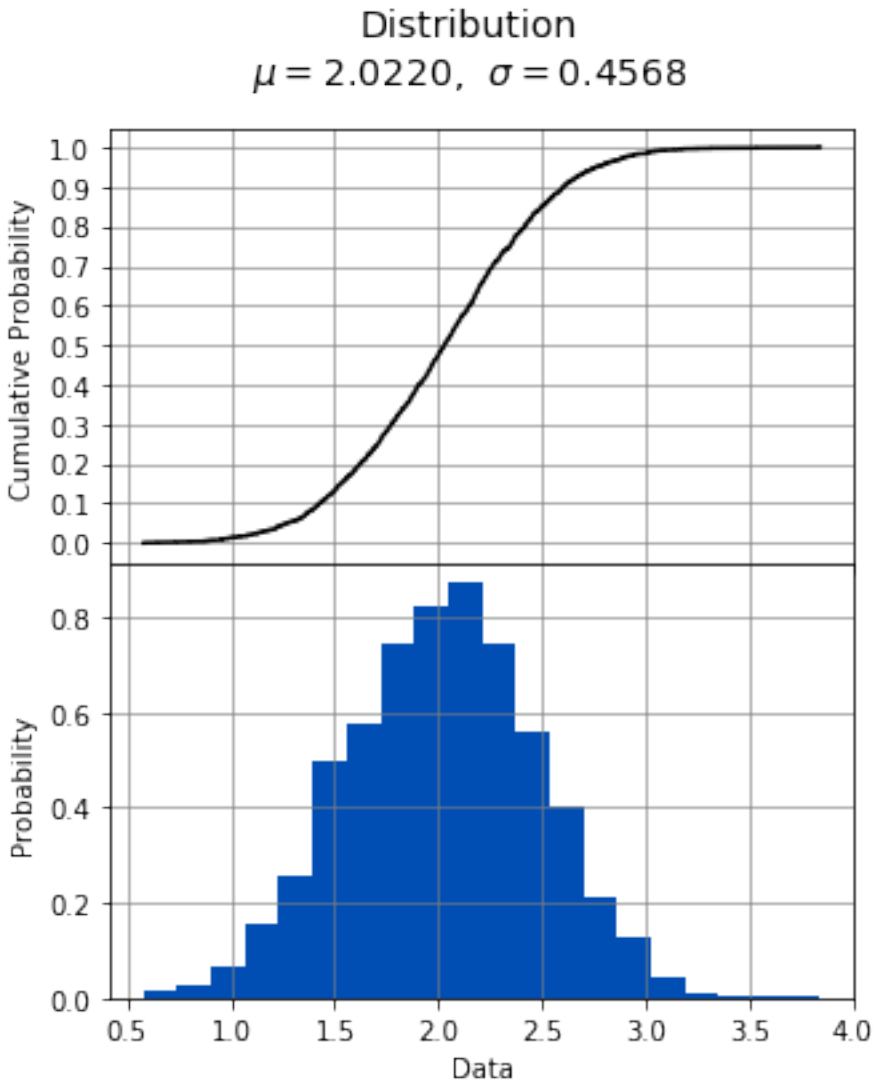
(continued from previous page)

```
IQR       =  0.6363
Range     =  3.2611


Shapiro-Wilk test for normality
-------------------------------

alpha   =  0.0500
W value =  0.9993
p value =  0.6430

H0: Data is normally distributed
```

Cumulative Distribution Function

The cumulative distribution function (cdf) differs from the probability density function in that it directly shows the probability of a data point occurring at a particular value in the distribution. The x-axis is the value in the distribution and the y-axis is the probability. For example, the center line of the y-axis is the 0.5 line (also known as the second quartile or Q2), and where the cdf crosses the 0.5 line is the median of the distribution.

```
analyze(
    sequence,
    boxplot=False,
    cdf=True,
)
```




```
Statistics
----------

n          =  2000
Mean       =  2.0220
Std Dev    =  0.4568
Std Error  =  0.0102
Skewness   = -0.0017
Kurtosis   = -0.0747
Maximum    =  3.8397
75%        =  2.3453
50%        =  2.0295
25%        =  1.7090
Minimum    =  0.5786
IQR        =  0.6363
Range      =  3.2611


Shapiro-Wilk test for normality
```

(continues on next page)

(continued from previous page)

```
------------------------------

alpha   =  0.0500
W value =  0.9993
p value =  0.6430

H0: Data is normally distributed
```

Interpreting the Statistics

There are two types of data that accompany the distribution analysis – the summary statistics and the test for normality.

Summary Statistics

- **n** - The number of data points in the analysis.
- **Mean** - The arithmetic mean or average of the analysis.
- **Std Dev** - The standard deviation of the analysis.
- **Std Error** - The standard error of the analysis.
- **Skewness** - A measure of how skewed the distribution is towards the min or max.
- **Kurtosis** - The kurtosis is a measure of how peaky the distribution is.
- **Maximum** - The largest value in the distribution.
- **75%** - The third quartile (Q3) of the distribution.
- **50%** - The second quartile (Q2) or median of the distribution.
- **25%** - The first quartile (Q1) of the distribution.
- **Minimum** - The smallest value in the distribution.
- **IQR** - The interquartile range of the distribution, which is calculated by Q3 - Q1.
- **Range** - The range of the distribution measured by Maximum - Minimum.

Test for normality

The Shapiro-Wilk test attempts to determine if the distribution closely resembles the normal distribution. If the p value is less than or equal to alpha, the distribution is considered to not be normally distributed.

Usage

analyze(*sequence*[, *box_plot=True*, *cdf=False*, *sample=False*, *bins=20*, *title='Distribution'*, *name='Data'*, *xname='Data'*, *yname='Probability'*, *save_to=None*])

Perform a Probability Distribution analysis on sequence.

Parameters

- **sequence** (*array-like*) – The array-like object to analyze. It can be a list, tuple, numpy array or pandas Series of numeric values.
- **boxplot** (*bool*) – Display the accompanying box plot if **True**.

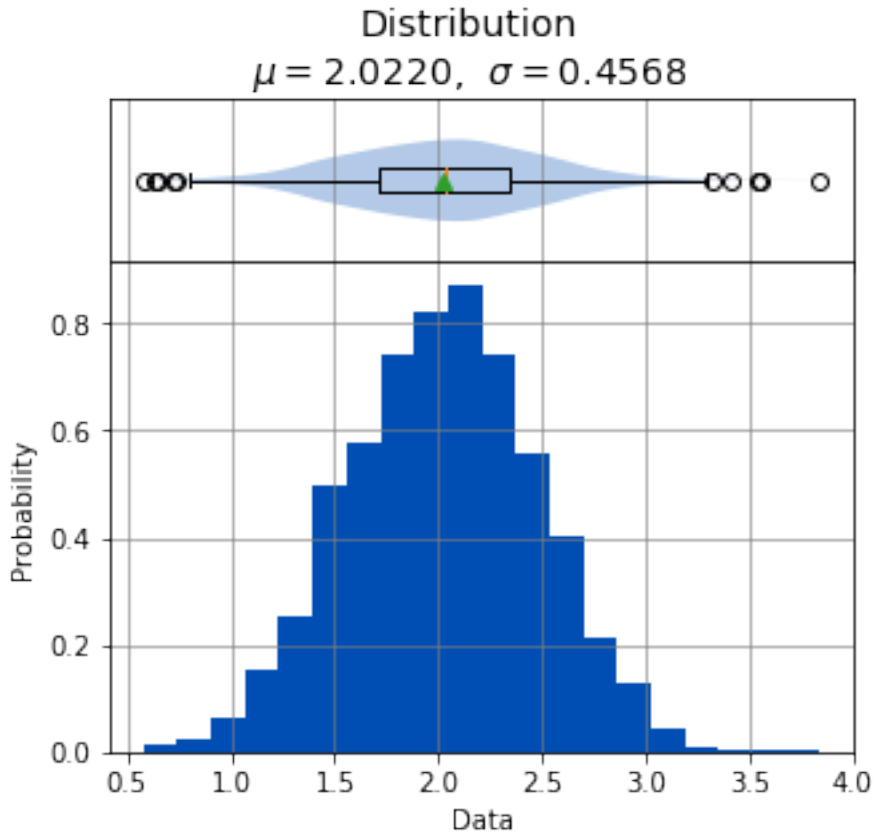
- **cdf** (*bool*) – Display the accompanying Cumulative Distribution Plot if **True**.
- **sample** (*bool*) – Sets x-bar and s if **True**, else mu and sigma when calculating summary statistics.
- **bins** (*int*) – The number of histogram bins to draw. The default value is 20.
- **title** (*str*) – The title of the graph.
- **name** (*str*) – The name of the distribution to show on the graph.
- **xname** (*str*) – Alias for name.
- **yname** (*str*) – The label of the y-axis of the histogram.

Argument Examples

sequence

The bare minimum requirement for performing a Distribution analysis. Should be an array-like of numeric values.

```
analyze(sequence)
```




```
Statistics
----------

n          =  2000
Mean       =  2.0220
```

(continues on next page)

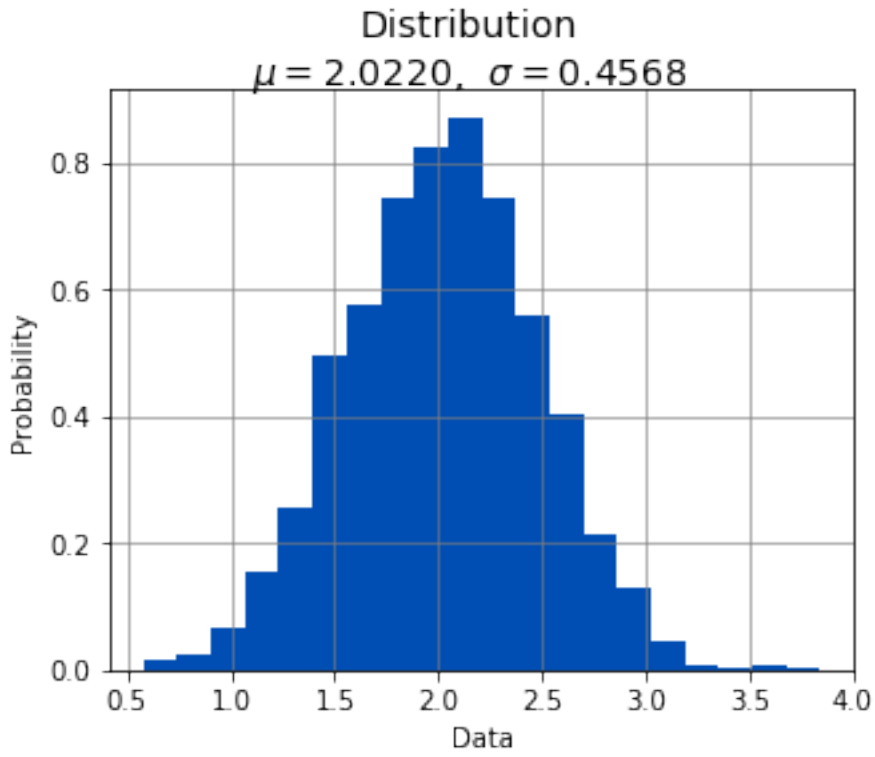
(continued from previous page)

```
Std Dev   =  0.4568
Std Error =  0.0102
Skewness  = -0.0017
Kurtosis  = -0.0747
Maximum   =  3.8397
75%       =  2.3453
50%       =  2.0295
25%       =  1.7090
Minimum   =  0.5786
IQR       =  0.6363
Range     =  3.2611


Shapiro-Wilk test for normality
-------------------------------

alpha   =  0.0500
W value =  0.9993
p value =  0.6430

H0: Data is normally distributed
```

boxplot

Controls whether the boxplot above the histogram is displayed or not. The default value is **True**.

```
analyze(
    sequence,
    boxplot=False,
)
```

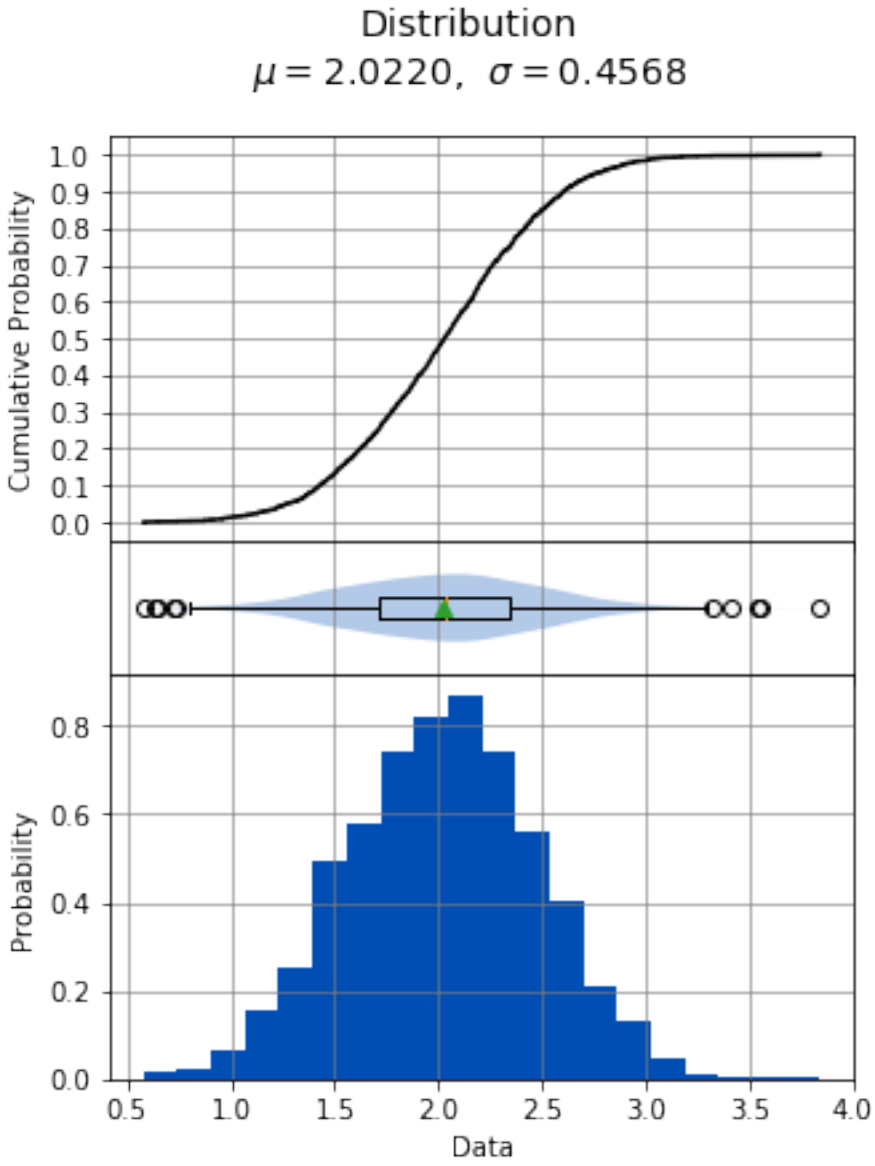



```
Statistics
----------

n          =  2000
Mean       =  2.0220
Std Dev    =  0.4568
Std Error  =  0.0102
Skewness   = -0.0017
Kurtosis   = -0.0747
Maximum    =  3.8397
75%        =  2.3453
50%        =  2.0295
25%        =  1.7090
Minimum    =  0.5786
IQR        =  0.6363
Range      =  3.2611


Shapiro-Wilk test for normality
-------------------------------

alpha    =  0.0500
W value  =  0.9993
p value  =  0.6430

H0: Data is normally distributed
```

cdf

Controls whether the cumulative distribution function is displayed or not. The default value is **False**.

```
analyze(
    sequence,
    cdf=True,
)
```




```
Statistics
----------

n         =  2000
Mean      =  2.0220
Std Dev   =  0.4568
Std Error =  0.0102
```

(continues on next page)

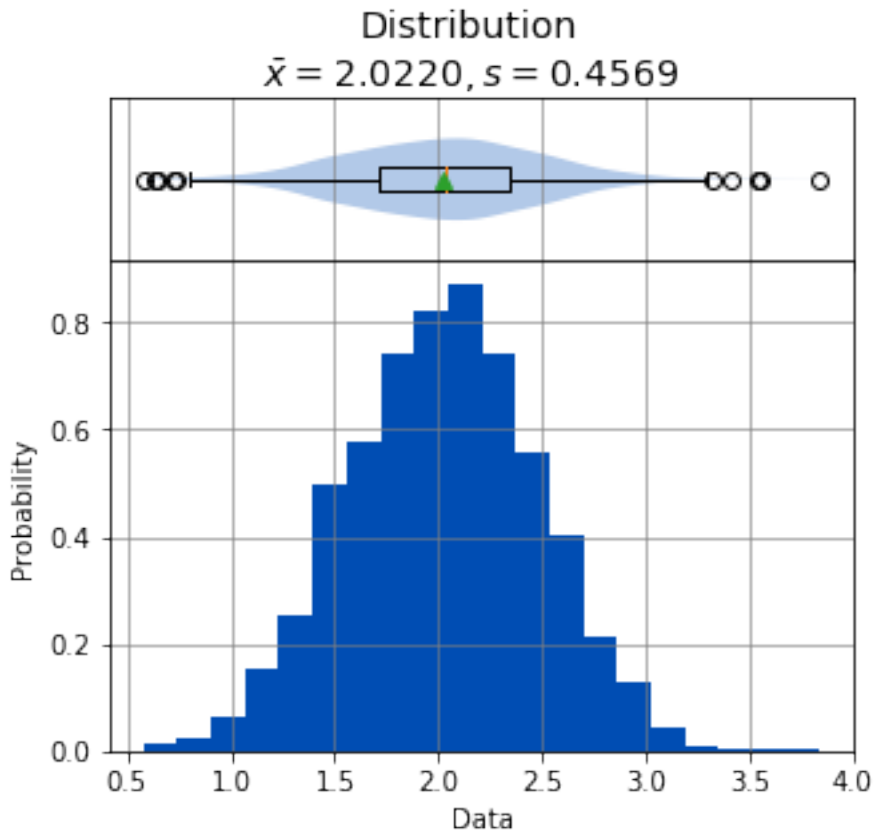
(continued from previous page)

```
Skewness  = -0.0017
Kurtosis  = -0.0747
Maximum   =  3.8397
75%       =  2.3453
50%       =  2.0295
25%       =  1.7090
Minimum   =  0.5786
IQR       =  0.6363
Range     =  3.2611


Shapiro-Wilk test for normality
-------------------------------

alpha   =  0.0500
W value =  0.9993
p value =  0.6430

H0: Data is normally distributed
```

sample

Controls whether the analysis is performed assuming whether *sequence* is a sample if **True**, or a population if **False**. The default value is **False**.

```
analyze(
    sequence,
    sample=True,
)
```



Statistics

```
n          =  2000
Mean       =  2.0220
Std Dev    =  0.4569
Std Error  =  0.0102
Skewness   = -0.0017
Kurtosis   = -0.0747
Maximum    =  3.8397
75%        =  2.3453
50%        =  2.0295
25%        =  1.7090
Minimum    =  0.5786
IQR        =  0.6363
Range      =  3.2611
```

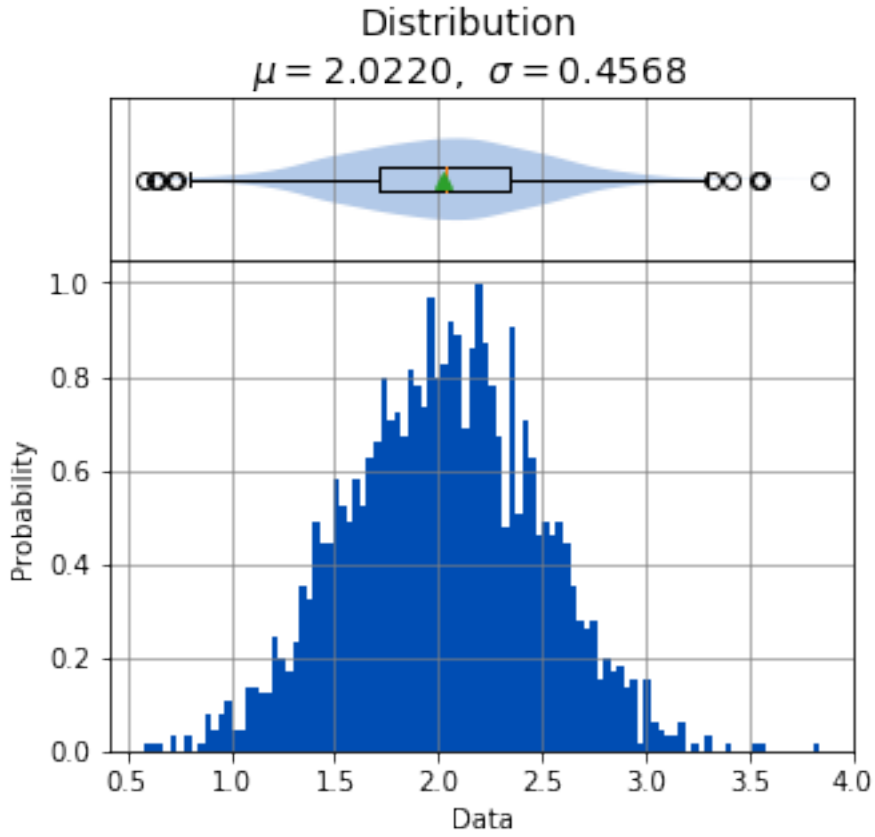
Shapiro-Wilk test for normality

```
alpha    =  0.0500
W value  =  0.9993
p value  =  0.6430

H0: Data is normally distributed
```

bins

Controls the number of bins to use for the histogram. The default value is 20.

```
analyze(
    sequence,
    bins=100,
)
```




```
Statistics
----------

n          =  2000
Mean       =  2.0220
Std Dev    =  0.4568
Std Error  =  0.0102
Skewness   = -0.0017
Kurtosis   = -0.0747
Maximum    =  3.8397
75%        =  2.3453
50%        =  2.0295
25%        =  1.7090
Minimum    =  0.5786
IQR        =  0.6363
Range      =  3.2611
```

(continues on next page)

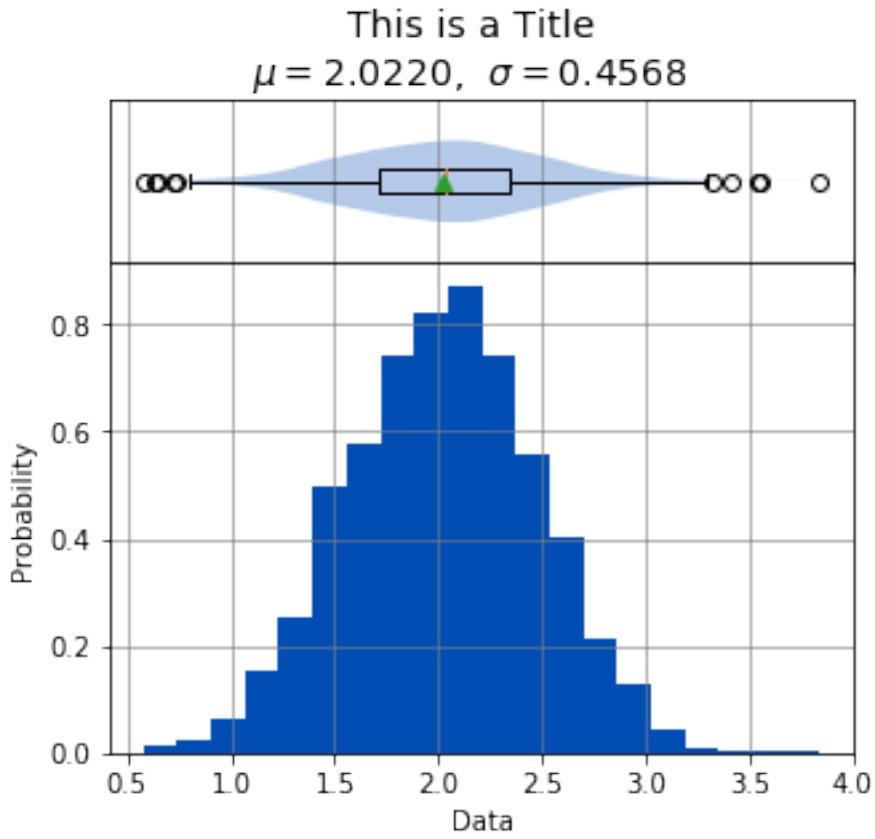
(continued from previous page)

```
Shapiro-Wilk test for normality
-------------------------------

alpha    =  0.0500
W value  =  0.9993
p value  =  0.6430

H0: Data is normally distributed
```

title

The title of the distribution to display above the graph.

```
analyze(
    sequence,
    title='This is a Title',
)
```




```
Statistics
----------

n          =  2000
Mean       =  2.0220
Std Dev    =  0.4568
```

(continues on next page)

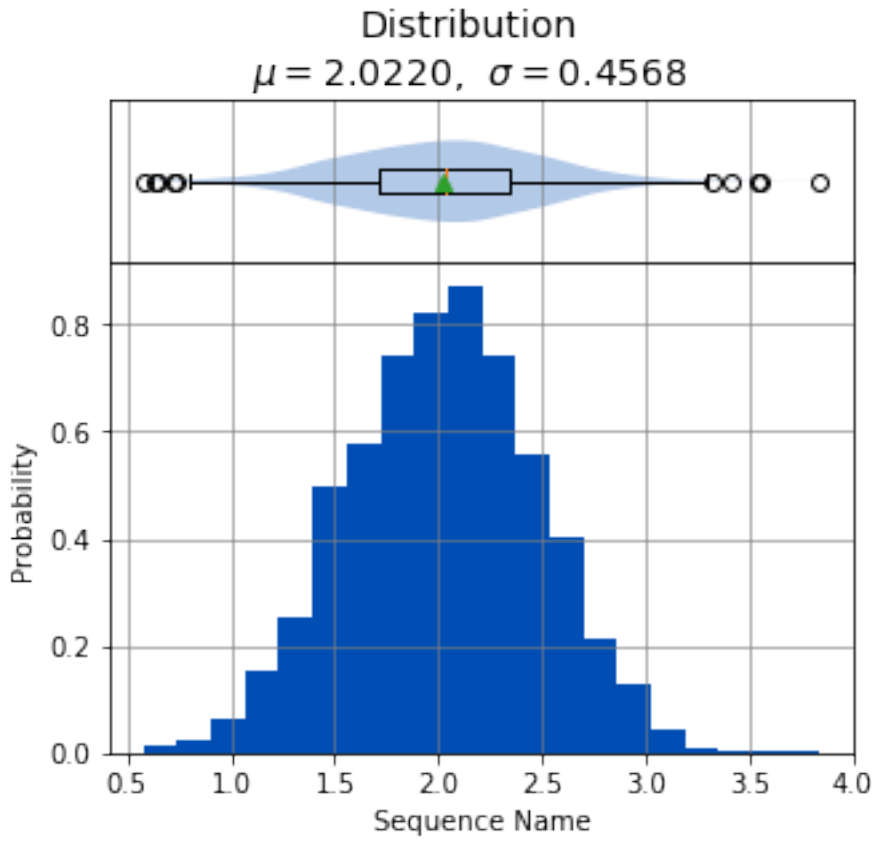
(continued from previous page)

```
Std Error =  0.0102
Skewness  = -0.0017
Kurtosis  = -0.0747
Maximum   =  3.8397
75%       =  2.3453
50%       =  2.0295
25%       =  1.7090
Minimum   =  0.5786
IQR       =  0.6363
Range     =  3.2611


Shapiro-Wilk test for normality
-------------------------------

alpha   =  0.0500
W value =  0.9993
p value =  0.6430

H0: Data is normally distributed
```

name, xname

The name of the distribution to display on the x-axis.

```
analyze(
    sequence,
    name='Sequence Name',
)
```



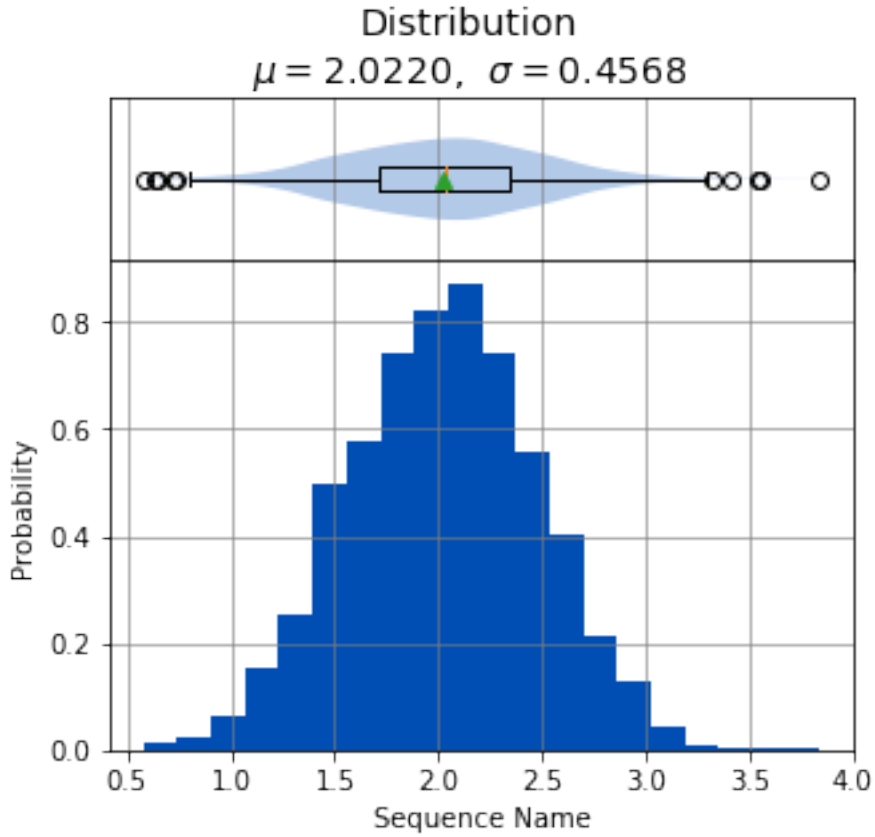
Statistics

```
n          =  2000
Mean       =  2.0220
Std Dev    =  0.4568
Std Error  =  0.0102
Skewness   = -0.0017
Kurtosis   = -0.0747
Maximum    =  3.8397
75%        =  2.3453
50%        =  2.0295
25%        =  1.7090
Minimum    =  0.5786
IQR        =  0.6363
Range      =  3.2611
```

Shapiro-Wilk test for normality

```
alpha    =  0.0500
W value  =  0.9993
p value  =  0.6430

H0: Data is normally distributed
```

```
analyze(
    sequence,
    xname='Sequence Name',
)
```




```
Statistics
----------

n          =  2000
Mean       =  2.0220
Std Dev    =  0.4568
Std Error  =  0.0102
Skewness   = -0.0017
Kurtosis   = -0.0747
Maximum    =  3.8397
75%        =  2.3453
50%        =  2.0295
25%        =  1.7090
Minimum    =  0.5786
IQR        =  0.6363
Range      =  3.2611


Shapiro-Wilk test for normality
-------------------------------

alpha    =  0.0500
```

(continues on next page)

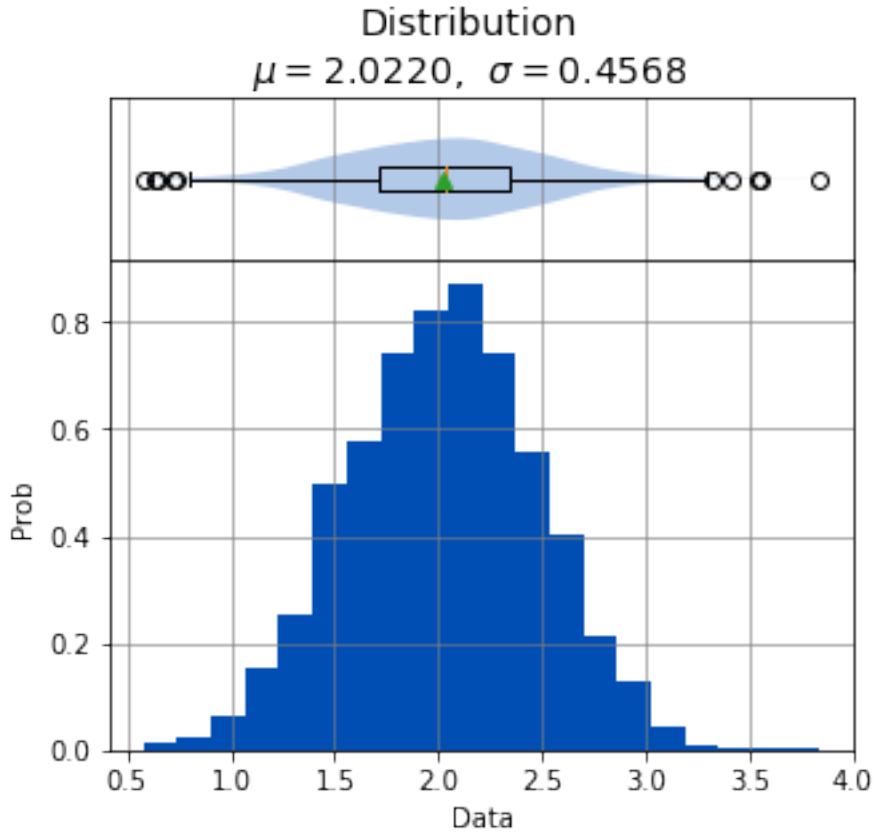
(continued from previous page)

```
W value =  0.9993
p value =  0.6430

H0: Data is normally distributed
```

yname

The value to display along the y-axis.

```
analyze(
    sequence,
    yname='Prob',
)
```




```
Statistics
----------

n          =  2000
Mean       =  2.0220
Std Dev    =  0.4568
Std Error  =  0.0102
Skewness   = -0.0017
Kurtosis   = -0.0747
Maximum    =  3.8397
```

(continues on next page)

(continued from previous page)

```
75%        =  2.3453
50%        =  2.0295
25%        =  1.7090
Minimum    =  0.5786
IQR        =  0.6363
Range      =  3.2611


Shapiro-Wilk test for normality
-------------------------------

alpha    =  0.0500
W value  =  0.9993
p value  =  0.6430

H0: Data is normally distributed
```

6.4.2 Frequency

Frequency analysis in sci-analysis is similar to Distribution analysis, but provides summary statistics and a bar chart of categorical data instead of numeric data. It provides the count, percent, and rank of the occurrence of each category in a given sequence.

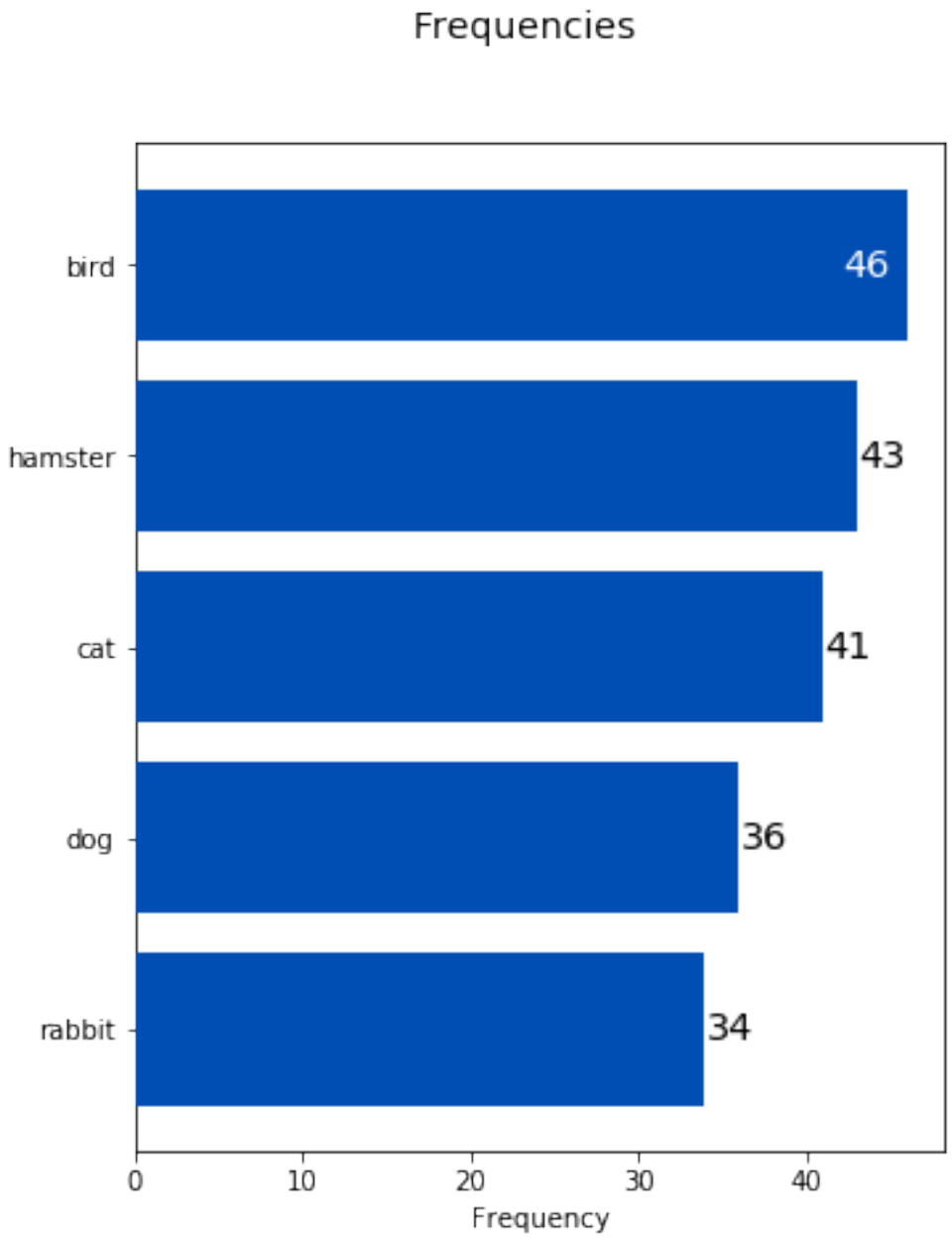
Interpreting the Graphs

The only graph shown by the frequency analysis is a bar chart where each bar is a unique category in the data set. By default the bar chart displays the frequency (counts) of each category in the bar chart, but can be configured to display the percent of each category instead.

```
import numpy as np
import scipy.stats as st
from sci_analysis import analyze

%matplotlib inline
```

```
np.random.seed(987654321)
pets = ['cat', 'dog', 'hamster', 'rabbit', 'bird']
sequence = [pets[np.random.randint(5)] for _ in range(200)]
```

```
analyze(sequence)
```




```
Overall Statistics
------------------

Total            =  200
Number of Groups =  5


Statistics
----------

Rank          Frequency     Percent        Category
-----------------------------------------------------------
1             46             23.0000       bird
```

(continues on next page)

(continued from previous page)

```
2               43              21.5000      hamster
3               41              20.5000      cat
4               36              18.0000      dog
5               34              17.0000      rabbit
```

Interpreting the Statistics

- **Total** - The total number of data points in the data set.
- **Number of Groups** - The number of unique categories in the data set.
- **Rank** - The ranking of largest category to smallest.
- **Frequency** - The number occurrences of each categorical value in the data set.
- **Percent** - The percent each category makes up of the entire data set.
- **Category** - The unique categorical values in the data set.

Usage

analyze(*sequence*[, *percent=False*, *vertical=True*, *grid=True*, *labels=True*, *dropna=False*, *order=None*, *title='Frequency'*, *name='Categories'*, *xname='Categories'*, *yname=None*, *save_to=None*])

Perform a Frequency analysis on *sequence*.

Parameters

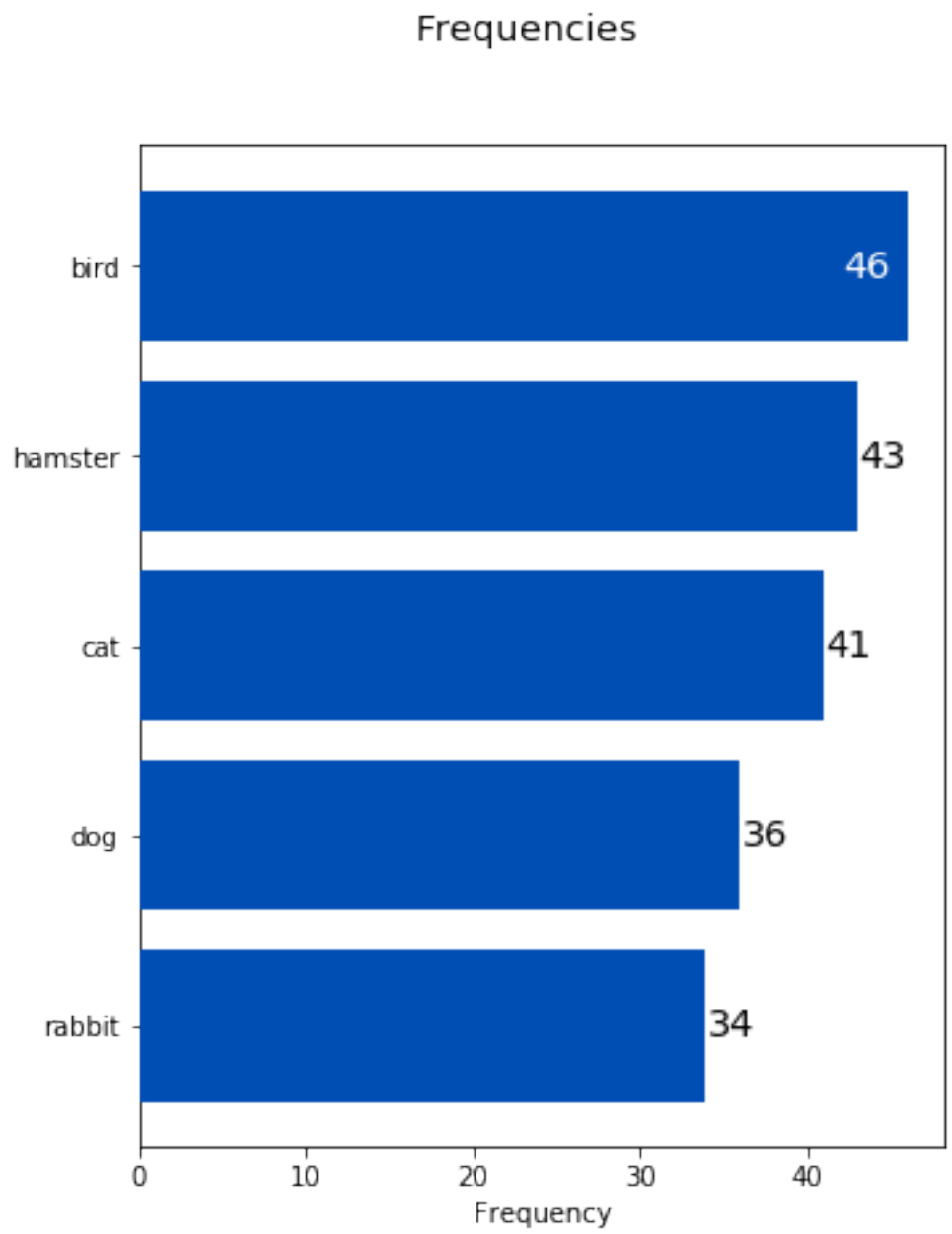
- **sequence** (*array-like*) – The array-like object to analyze. It can be a list, tuple, numpy array or pandas Series of string values.
- **percent** (*bool*) – Display the percent of each category on the bar chart if **True**, otherwise will display the count of each category.
- **vertical** (*bool*) – Display the bar chart with a vertical orientation if **True**.
- **grid** (*bool*) – Add grid lines to the bar chart if **True**.
- **labels** (*bool*) – Display count or percent labels on the bar chart for each group if **True**.
- **dropna** (*bool*) – If **False**, missing values in sequence are grouped together as their own category on the bar chart.
- **order** (*array-like*) – Sets the order of the categories displayed on the bar chart according to the order of values in *order*.
- **title** (*str*) – The title of the graph.
- **name** (*str*) – The name of the data to show on the graph.
- **xname** (*str*) – Alias for name.
- **yname** (*str*) – The label of the y-axis of the bar chart. The default is "Percent" if percent is **True**, otherwise the default is "Count."
- **save_to** (*str*) – If a string value, the path to save the graph to.

Argument Examples

sequence

A sequence of categorical values to be analyzed.

```
analyze(sequence)
```




```
Overall Statistics
------------------

Total            =  200
Number of Groups =  5
```

(continues on next page)

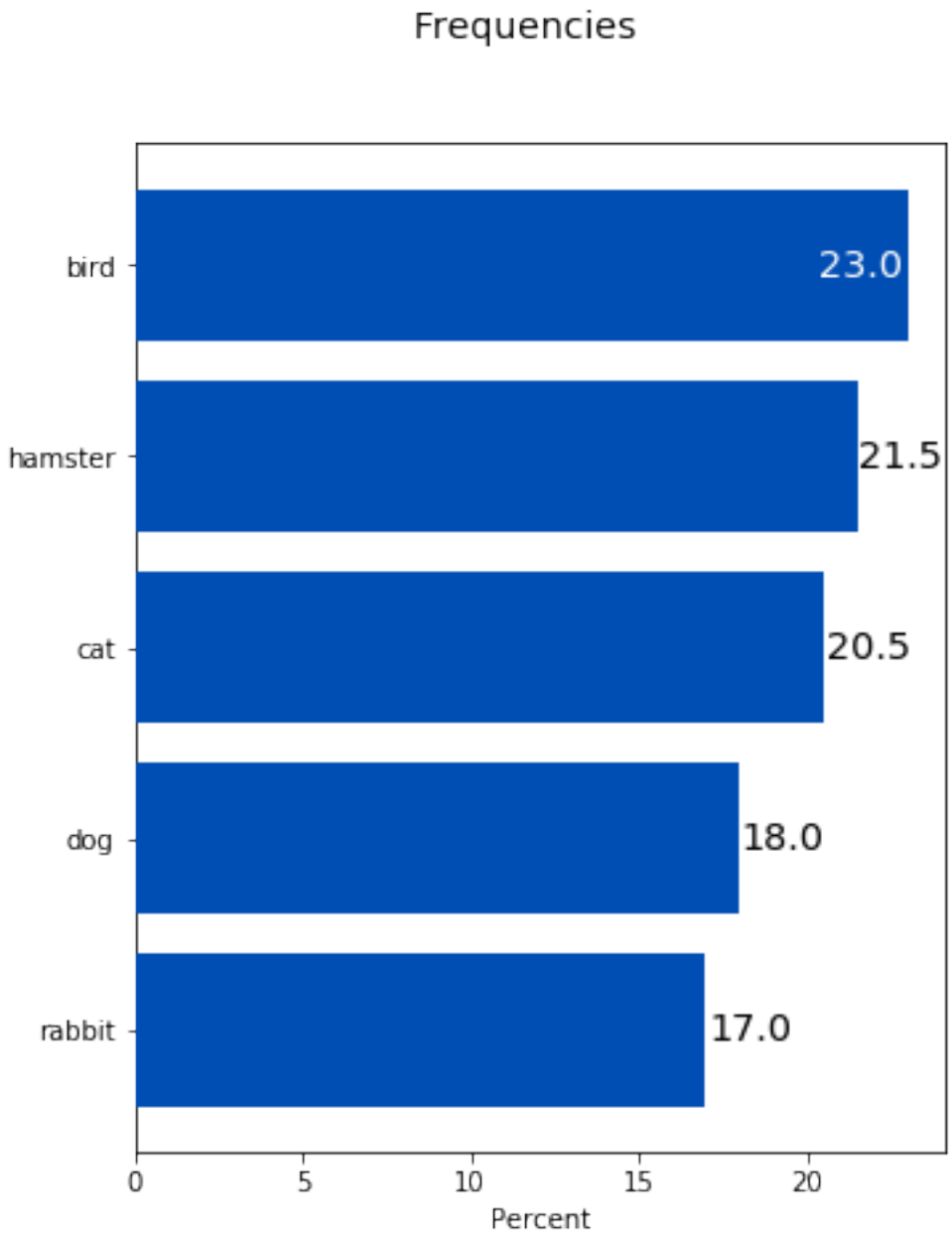
(continued from previous page)

```
Statistics
----------

Rank          Frequency     Percent       Category
-----------------------------------------------------
1             46             23.0000      bird
2             43             21.5000      hamster
3             41             20.5000      cat
4             36             18.0000      dog
5             34             17.0000      rabbit
```

percent

Controls whether percents are displayed instead of counts on the bar chart. The default is **False**.

```
analyze(
    sequence,
    percent=True,
)
```




```
Overall Statistics
------------------

Total            =  200
Number of Groups =  5


Statistics
----------

Rank          Frequency      Percent        Category
-----------------------------------------------------------
1             46              23.0000       bird
```

(continues on next page)

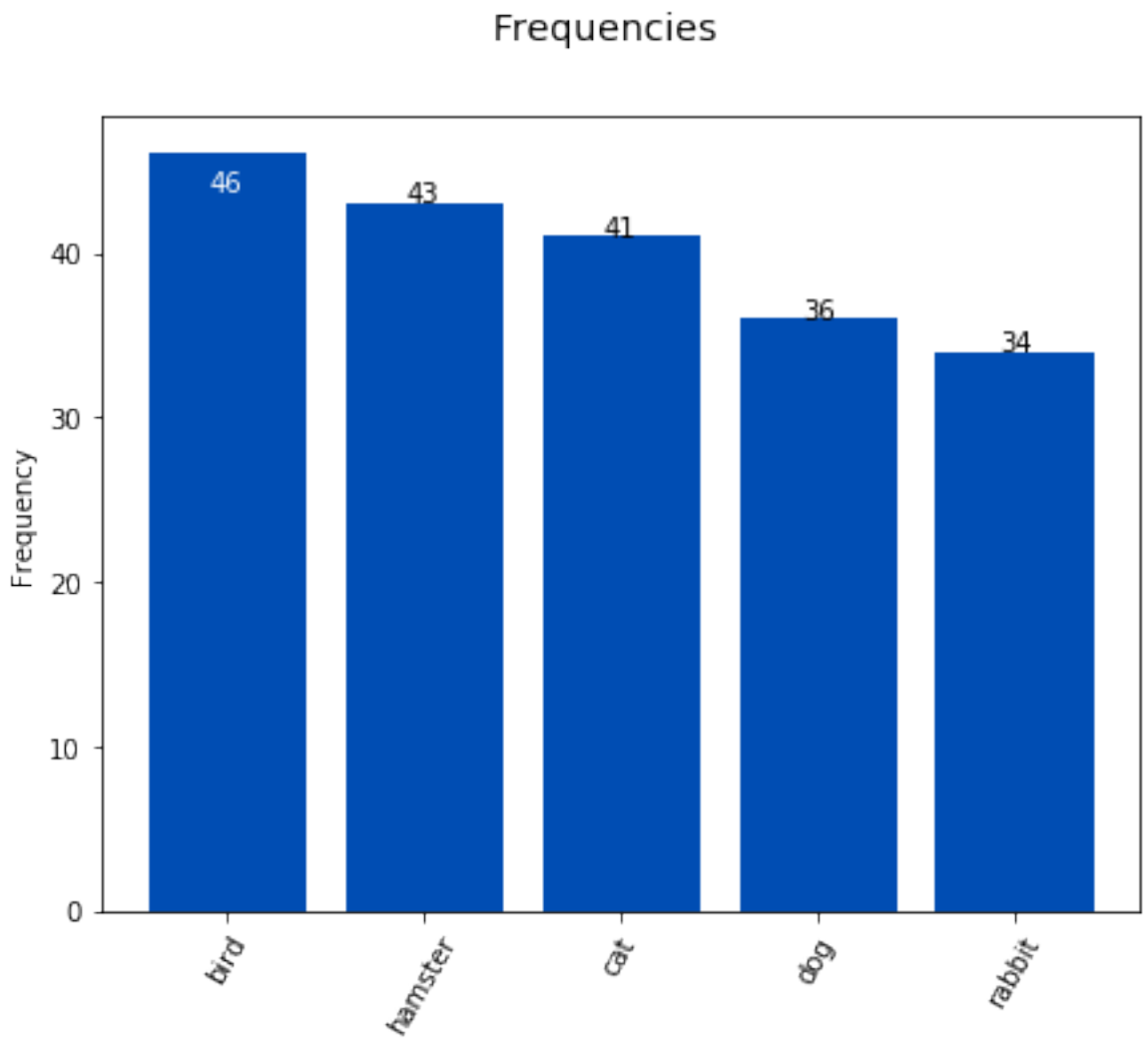
(continued from previous page)

```
2               43              21.5000       hamster
3               41              20.5000       cat
4               36              18.0000       dog
5               34              17.0000       rabbit
```

vertical

Controls whether the bar chart is displayed in a vertical orientation or not. The default is **True**.

```
analyze(
    sequence,
    vertical=False,
)
```




```
Overall Statistics
------------------
```

(continues on next page)

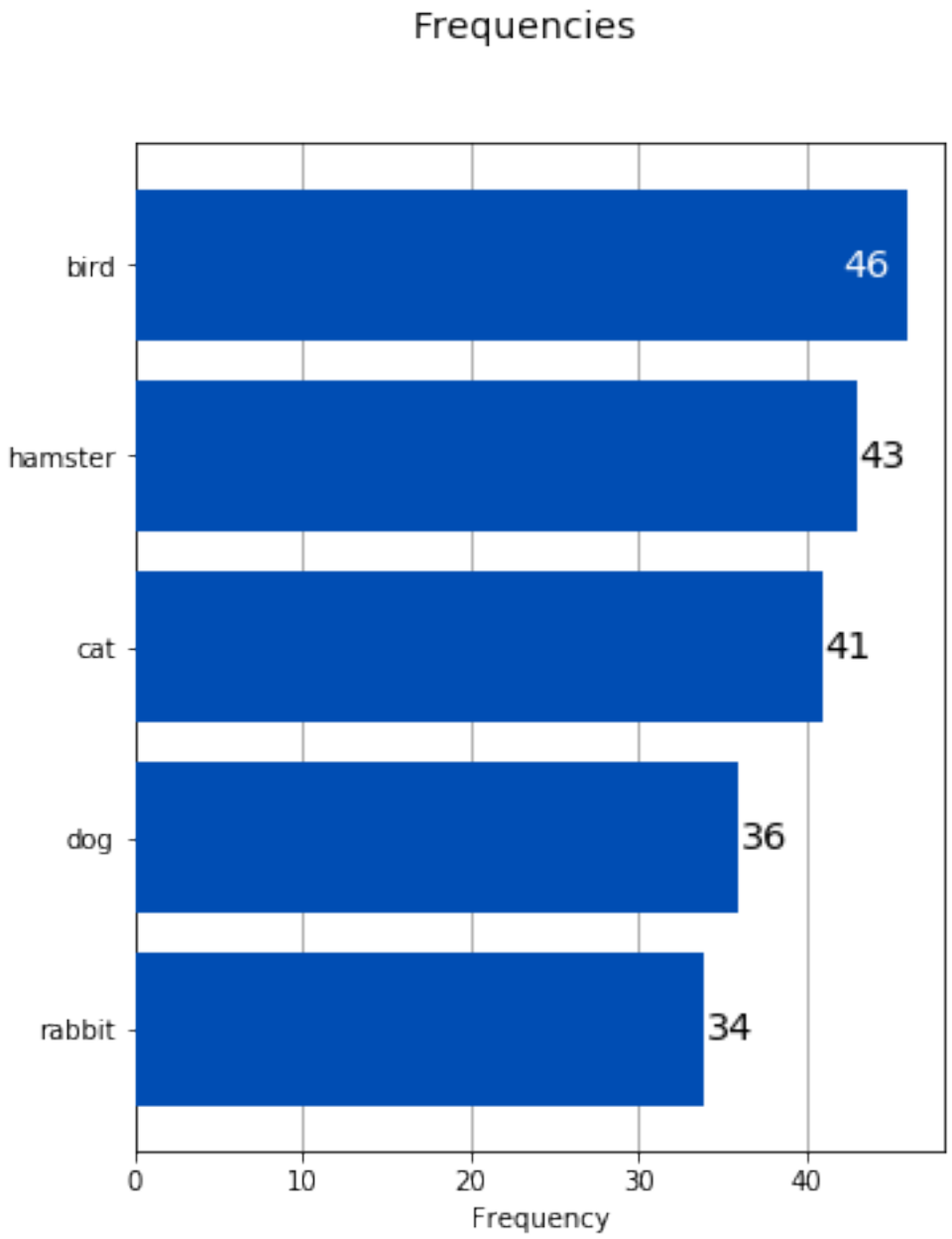
(continued from previous page)

```
Total            =  200
Number of Groups =  5


Statistics
----------

Rank          Frequency     Percent       Category
-------------------------------------------------------
1             46             23.0000      bird
2             43             21.5000      hamster
3             41             20.5000      cat
4             36             18.0000      dog
5             34             17.0000      rabbit
```

grid

Controls whether the grid is displayed on the bar chart or not. The default is **False**.

```
analyze(
    sequence,
    grid=True,
)
```




```
Overall Statistics
------------------

Total            =  200
Number of Groups =  5


Statistics
----------

Rank          Frequency     Percent        Category
-----------------------------------------------------------
1             46             23.0000       bird
```

(continues on next page)

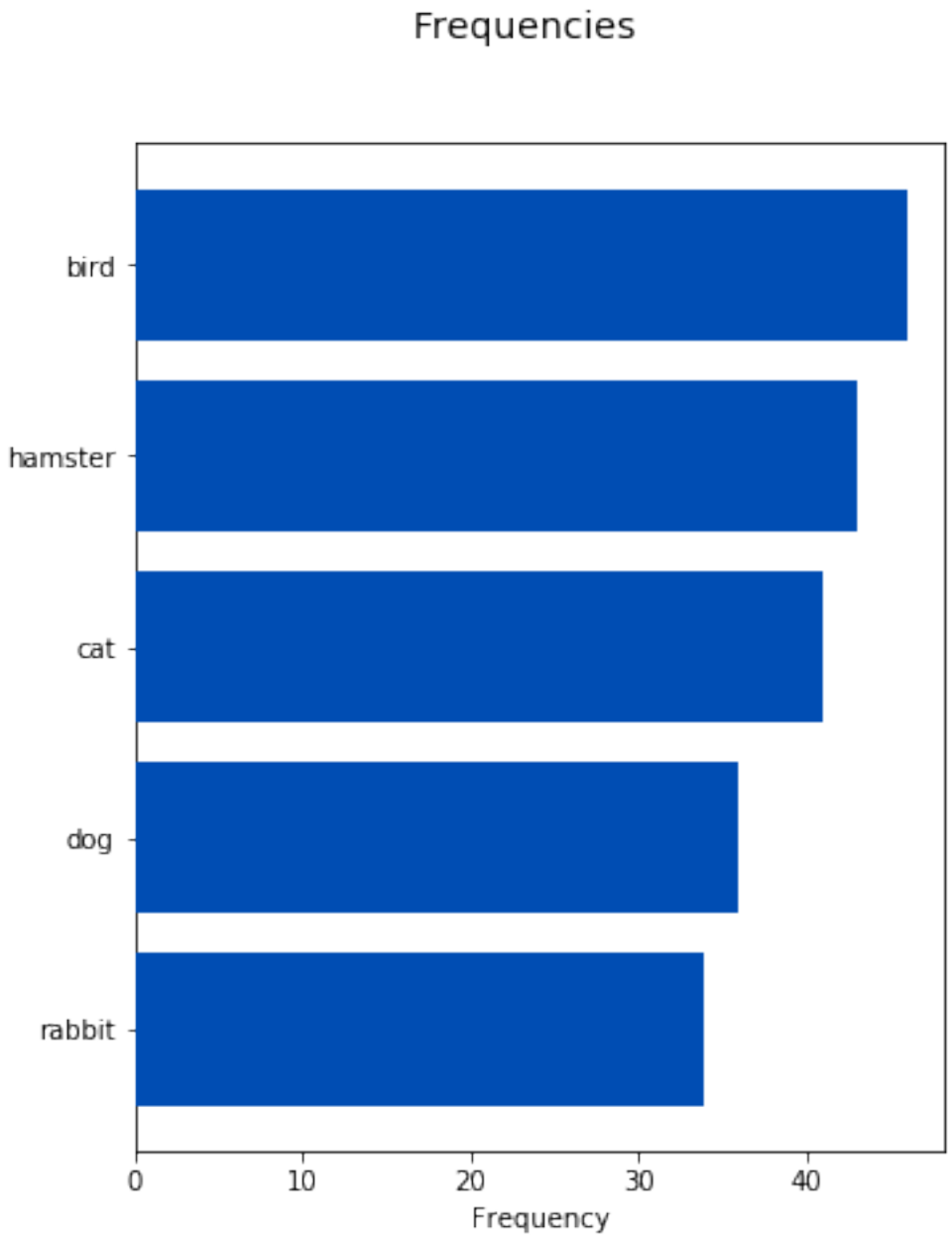
(continued from previous page)

```
2               43              21.5000       hamster
3               41              20.5000       cat
4               36              18.0000       dog
5               34              17.0000       rabbit
```

labels

Controls whether the count or percent labels are displayed or not. The default is **True**.

```
analyze(
    sequence,
    labels=False,
)
```




```
Overall Statistics
------------------

Total            =  200
Number of Groups =  5


Statistics
----------

Rank          Frequency     Percent        Category
-----------------------------------------------------------
1             46             23.0000       bird
```

(continues on next page)

(continued from previous page)

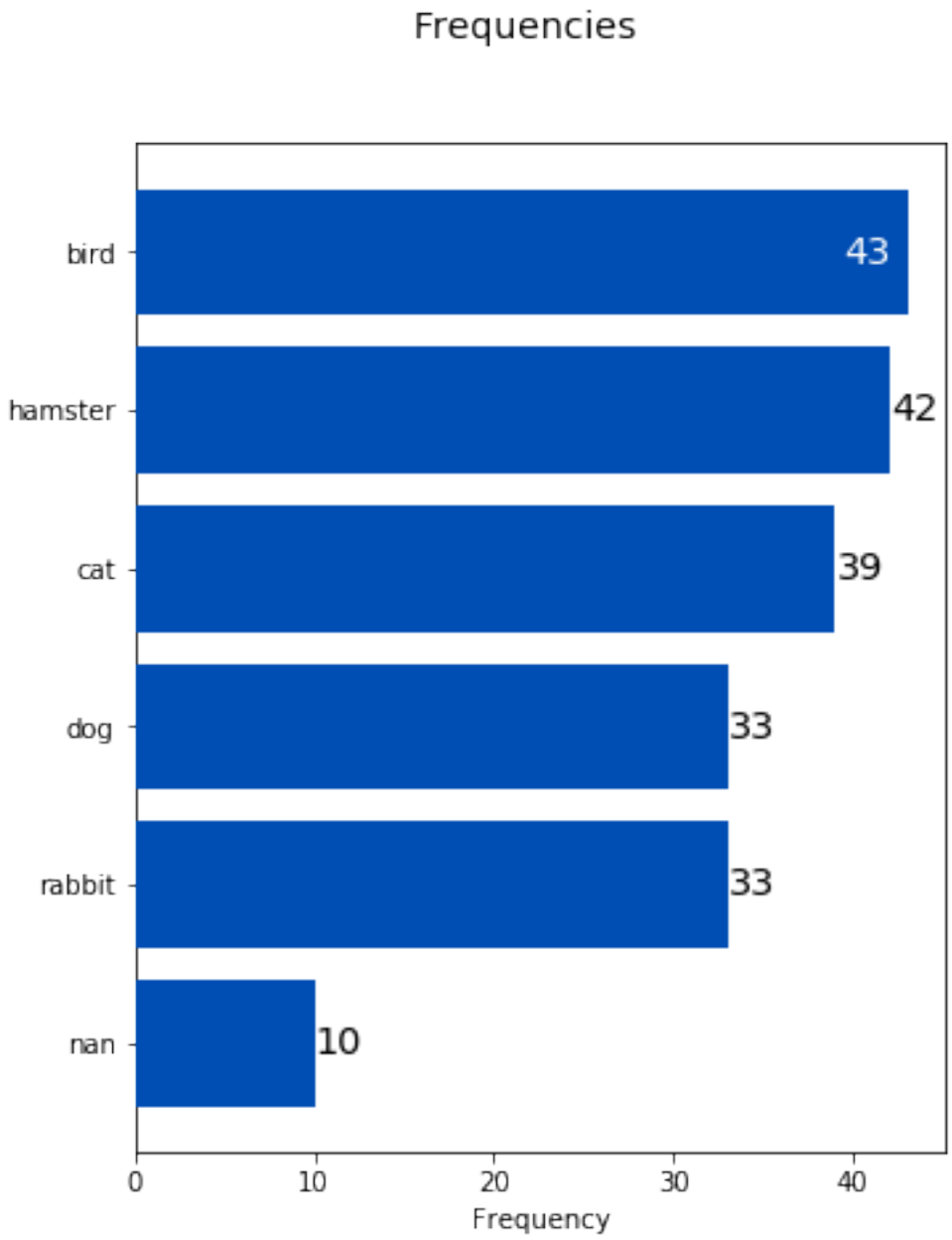
```
2               43              21.5000       hamster
3               41              20.5000       cat
4               36              18.0000       dog
5               34              17.0000       rabbit
```

dropna

Removes missing values from the bar chart if **True**, otherwise, missing values are grouped together into a category called "nan". The default is **False**.

```
# Convert 10 random values in sequence to NaN.
for _ in range(10):
    sequence[np.random.randint(200)] = np.nan
```

```
analyze(sequence)
```



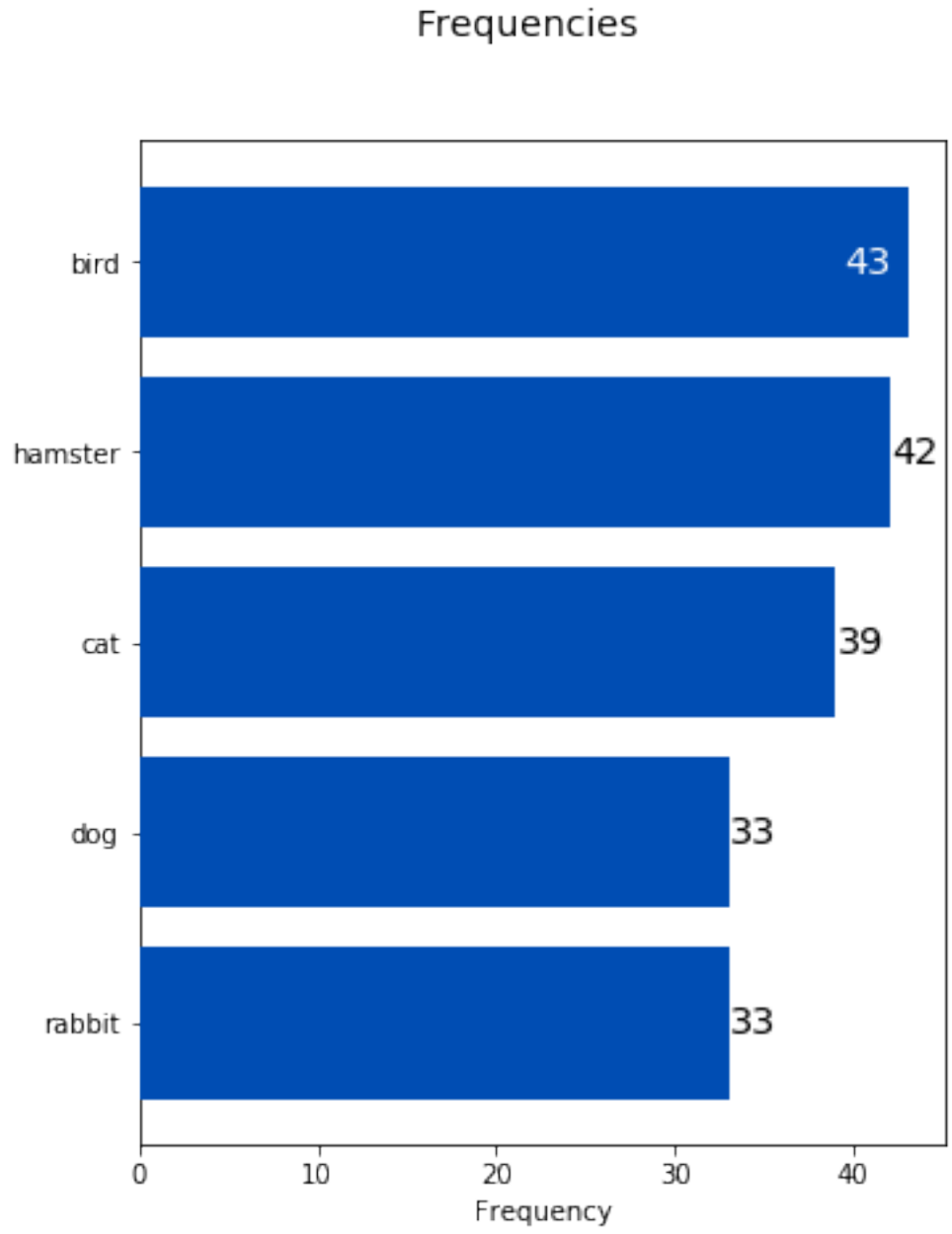

```
Overall Statistics
------------------

Total            =  200
Number of Groups =  6


Statistics
----------

Rank          Frequency      Percent        Category
-------------------------------------------------------
1             43              21.5000       bird
```

(continues on next page)

(continued from previous page)

```
2               42              21.0000       hamster
3               39              19.5000       cat
4               33              16.5000       dog
4               33              16.5000       rabbit
5               10              5.0000         nan
```

```
analyze(
    sequence,
    dropna=True,
)
```

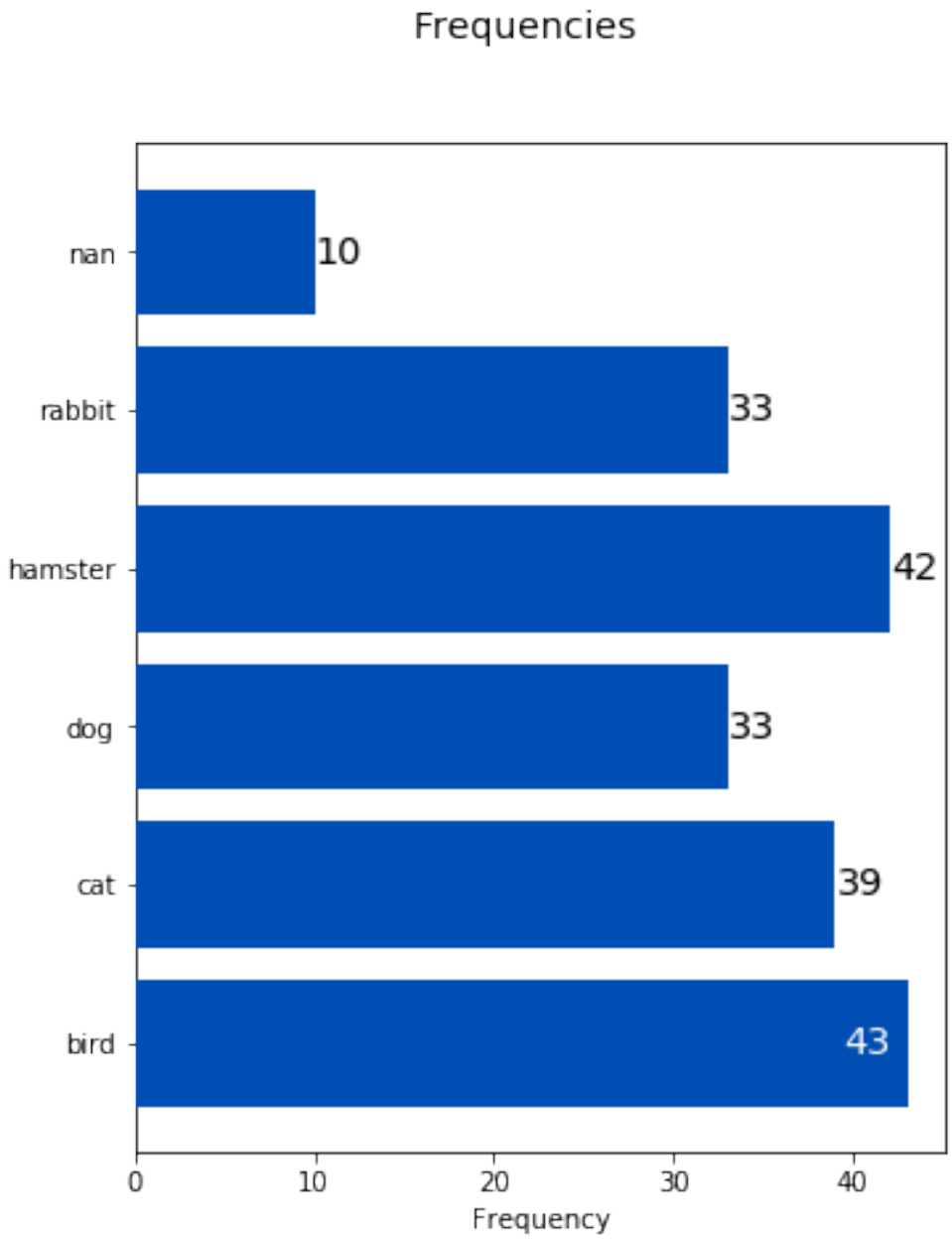


```
Overall Statistics
------------------

Total            =  200
Number of Groups =  6


Statistics
----------

Rank          Frequency     Percent       Category
-----------------------------------------------------------
1             43             21.5000      bird
2             42             21.0000      hamster
3             39             19.5000      cat
4             33             16.5000      dog
4             33             16.5000      rabbit
5             10             5.0000        nan
```

order

A list of category names that sets the order for how categories are displayed on the bar chart. If sequence contains missing values, the category "nan" is shown first.

```
analyze(
    sequence,
    order=['rabbit', 'hamster', 'dog', 'cat', 'bird'],
)
```




```
Overall Statistics
------------------

Total            =  200
Number of Groups =  6


Statistics
----------

Rank          Frequency     Percent       Category
-----------------------------------------------------
1             43             21.5000      bird
```

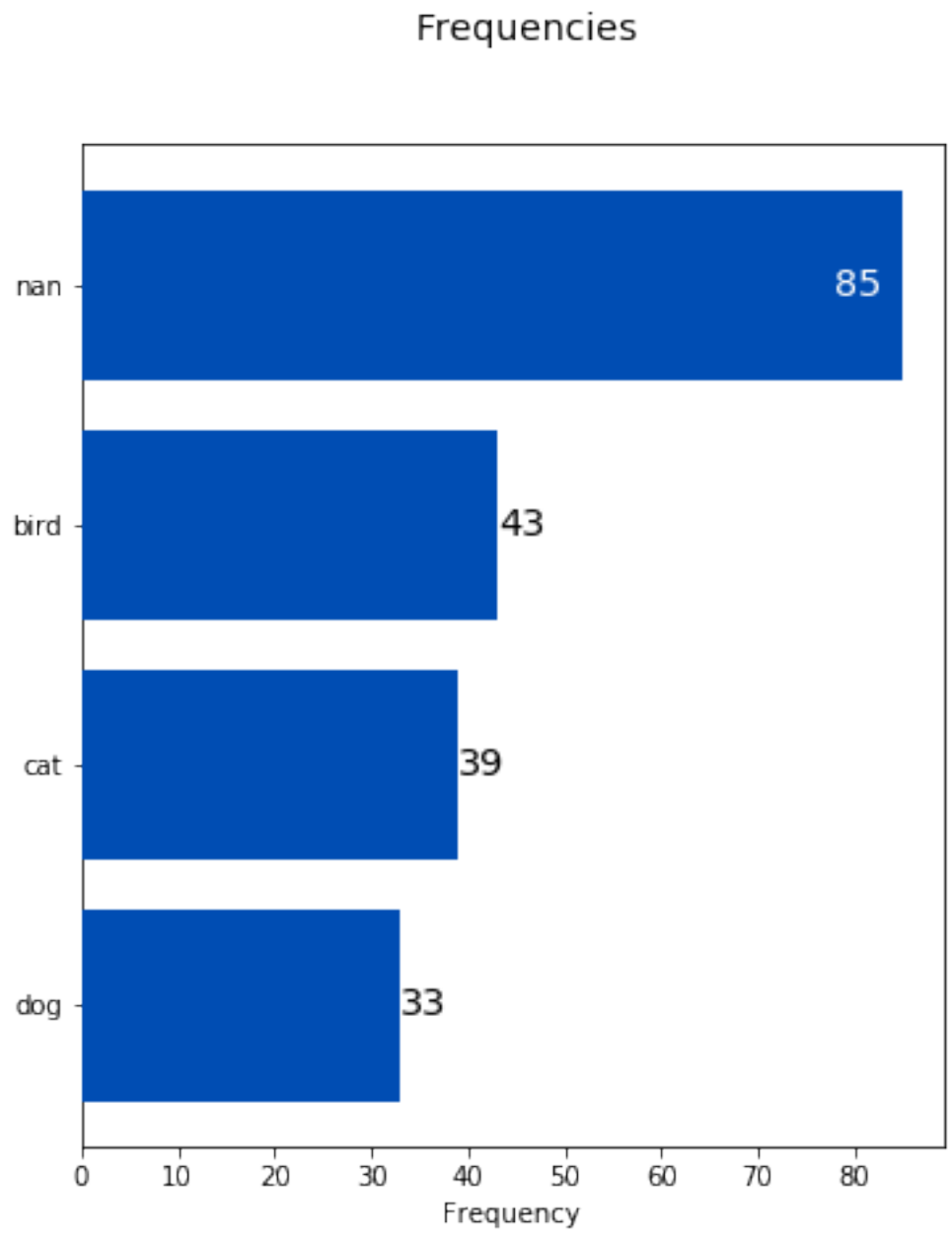
(continues on next page)

(continued from previous page)

```
2               42              21.0000       hamster
3               39              19.5000       cat
4               33              16.5000       dog
4               33              16.5000       rabbit
5               10              5.0000         nan
```

If there are categories in *sequence* that aren't listed in *order*, they are reported as "nan" on the bar chart.

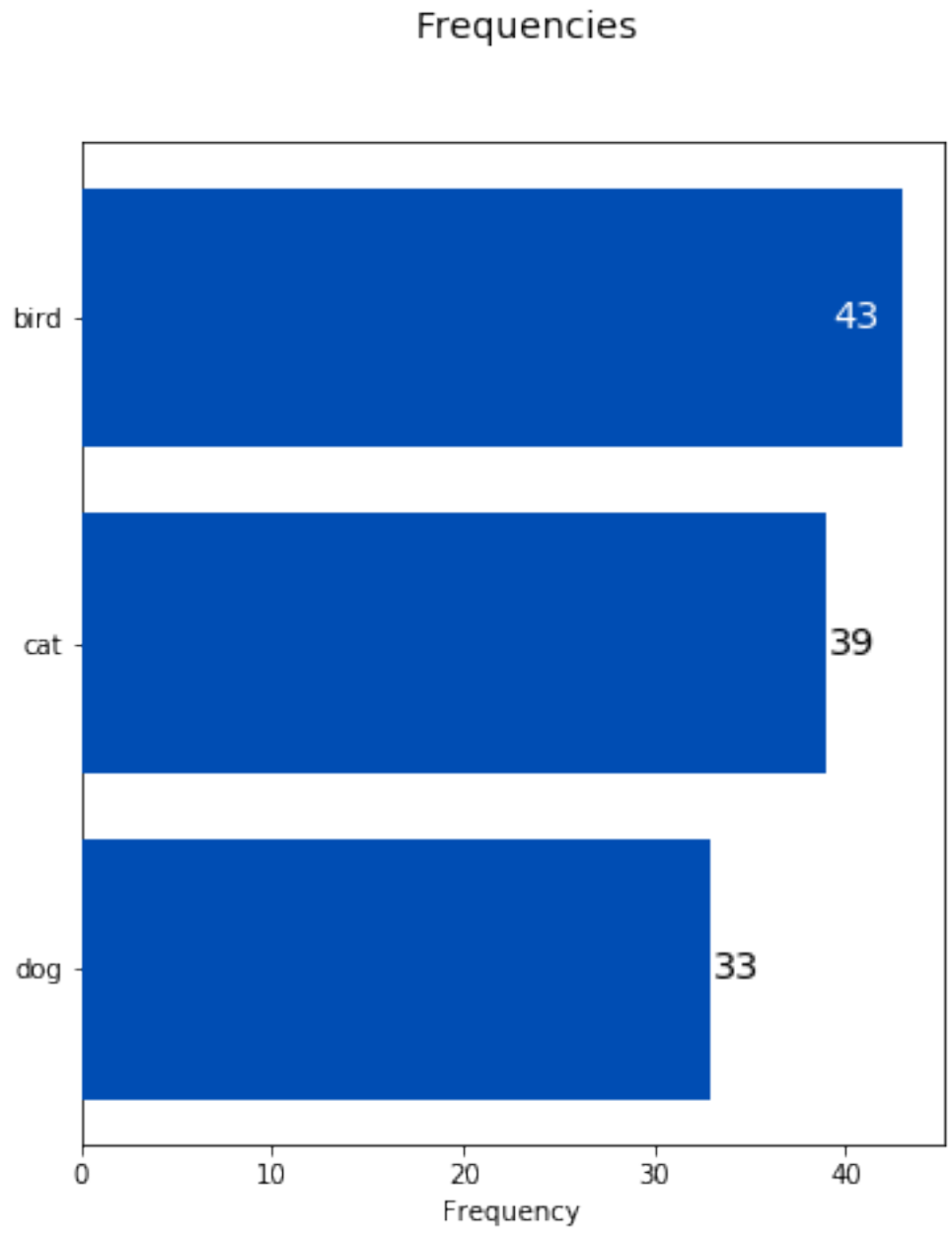
```
analyze(
    sequence,
    order=['bird', 'cat', 'dog'],
)
```



```
Overall Statistics
------------------

Total            =  200
Number of Groups =  6


Statistics
----------

Rank          Frequency     Percent       Category
-----------------------------------------------------------
1             43             21.5000      bird
2             42             21.0000      hamster
3             39             19.5000      cat
4             33             16.5000      dog
4             33             16.5000      rabbit
5             10             5.0000        nan
```

Missing values can be dropped from the bar chart with *dropna*=**True**.

```
analyze(
    sequence,
    order=['bird', 'cat', 'dog'],
    dropna=True,
)
```




```
Overall Statistics
------------------

Total            =  200
Number of Groups =  6


Statistics
----------

Rank          Frequency     Percent       Category
-----------------------------------------------------------
1             43             21.5000      bird
```

(continues on next page)

(continued from previous page)

```
2              42             21.0000      hamster
3              39             19.5000      cat
4              33             16.5000      dog
4              33             16.5000      rabbit
5              10             5.0000        nan
```

6.4.3 Bivariate

A bivariate analysis differs from a univariate, or distribution analysis, in that it is the analysis of two separate sets of data. These two sets of data are compared to one another to check for correlation, or a tendency of one of the sets of data to "predict" corresponding values in the other data set. If a linear or higher order model can be applied to describe, or model, the two sets of data, they are said to be correlated.

When two distributions are correlated, it is possible that the data in one of the distributions can be used to predict a corresponding value in the second distribution. This first distribution is referred to as the predictor and the second distribution as the response. Both predictor and response are graphed by a scatter plot, typically with the predictor on the x-axis and the response on the y-axis.

Note: Just because two sets of data correlate with one another does not necessarily mean that one predicts the other. It merely means it's a possibility that one predicts the other. This is summarized by the saying "Correlation does not imply causation." Use caution when drawing conclusions of a bivariate analysis. It is a good idea to study both data sets more carefully to determine if the two data sets are in fact correlated.

Interpreting the Graphs

Let's first import sci-analysis and setup some variables to use in these examples.

```
import numpy as np
import scipy.stats as st
from sci_analysis import analyze

%matplotlib inline
```

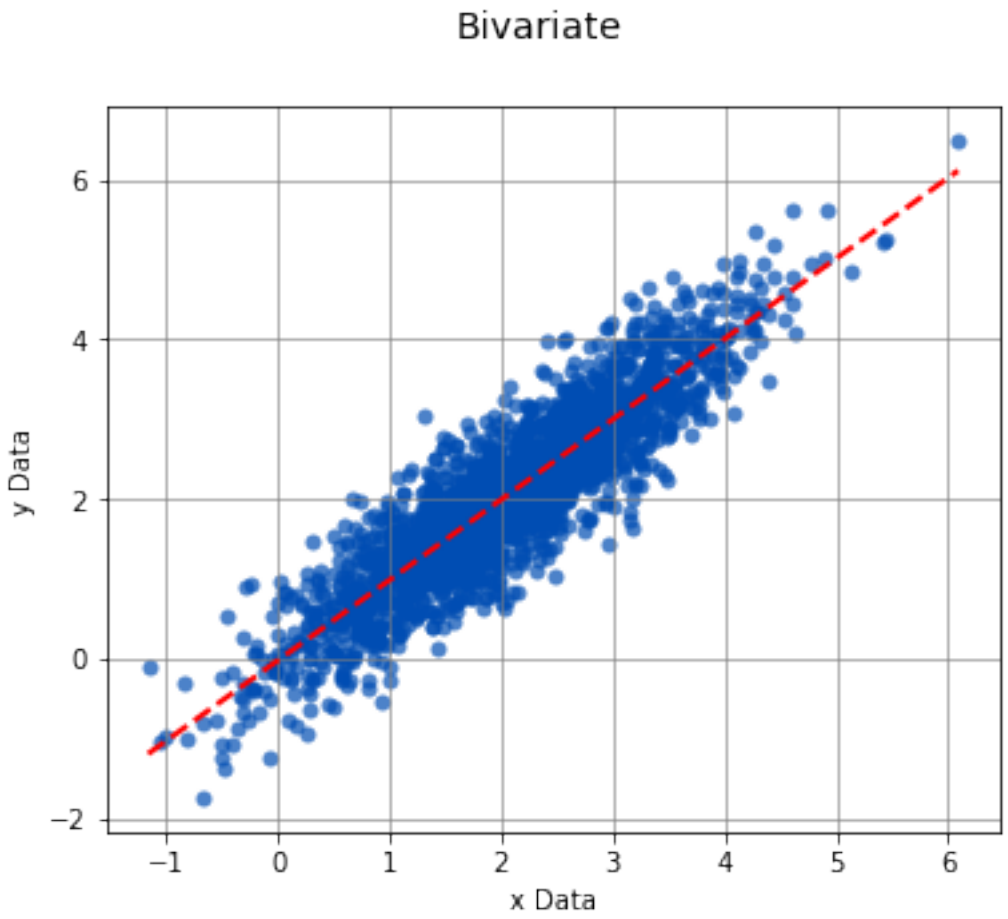
```
# Create x-sequence and y-sequence from random variables.
np.random.seed(987654321)
x_sequence = st.norm.rvs(2, size=2000)
y_sequence = np.array([x + st.norm.rvs(0, 0.5, size=1) for x in x_sequence])
```

Scatter Plot

A scatter plot is used in sci-analysis to visualize the correlation between two sets of data. For this to work, each value in the first set of data has a corresponding value in the second set of data. The two values are tied together by the matching index value in each set of data. The length of each set of data have to be equal to one another, and the index values of each data set have to contiguous. If there is a missing value or values in one data set, the matching value at the same index in the other data set will be dropped.

By default, the best-fit line (assuming a linear relationship) is drawn as a dotted red line.

```
analyze(x_sequence, y_sequence)
```

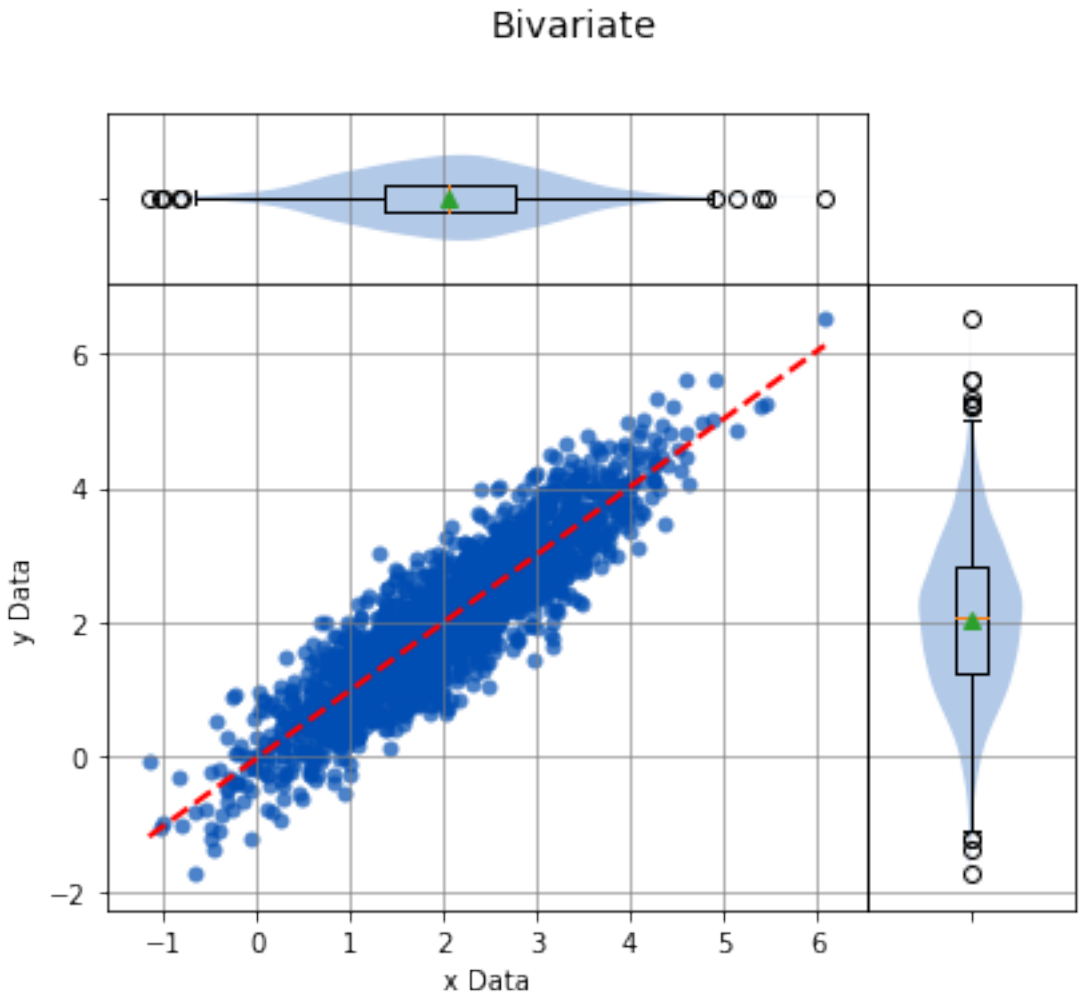



```
Linear Regression
-----------------

n          =  2000
Slope      =  1.0059
Intercept = -0.0133
r          =  0.8974
r^2        =  0.8054
Std Err    =  0.0111
p value    =  0.0000



Pearson Correlation Coefficient
-------------------------------

alpha   =  0.0500
r value =  0.8974
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

Boxplot Borders

Boxplots can be displayed along-side the x and y axis of the scatter plot. This is a useful tool for visualizing the distribution of the sets of data on the x and y axis while still displaying the scatter plot.

```
analyze(x_sequence, y_sequence, boxplot_borders=True)
```




```
Linear Regression
-----------------

n         =  2000
Slope     =  1.0059
Intercept = -0.0133
r         =  0.8974
r^2       =  0.8054
Std Err   =  0.0111
p value   =  0.0000



Pearson Correlation Coefficient
```

(continues on next page)

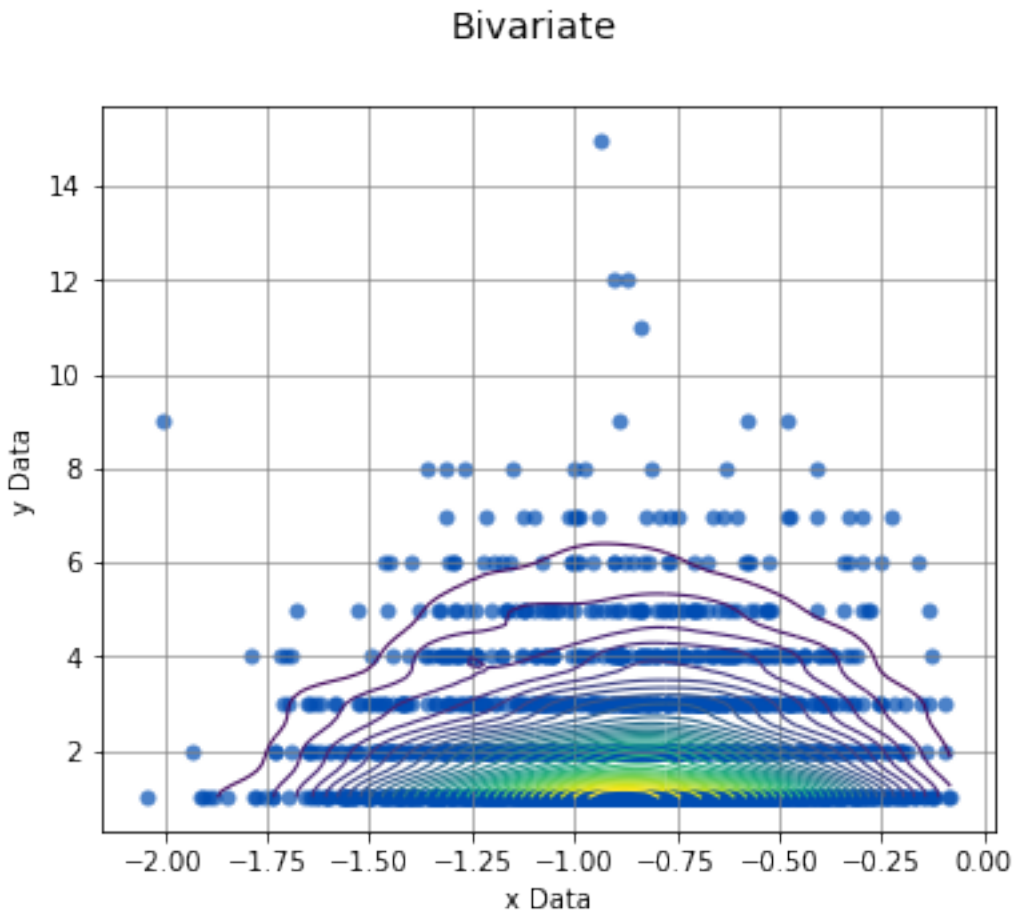
(continued from previous page)

```
------------------------------

alpha   =  0.0500
r value =  0.8974
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

Contours

In certain cases, such as when one of the sets of data is discrete and the other is continuous, it might be difficult to determine where the data points are centered. In this case, density contours can be used to help visualize the join probability distribution between the two sets of data.

```
x_continuous = st.weibull_max.rvs(2.7, size=2000)
y_discrete = st.geom.rvs(0.5, loc=0, size=2000)
analyze(x_continuous, y_discrete, contours=True, fit=False)
```




```
Linear Regression
-----------------
```

(continues on next page)

(continued from previous page)

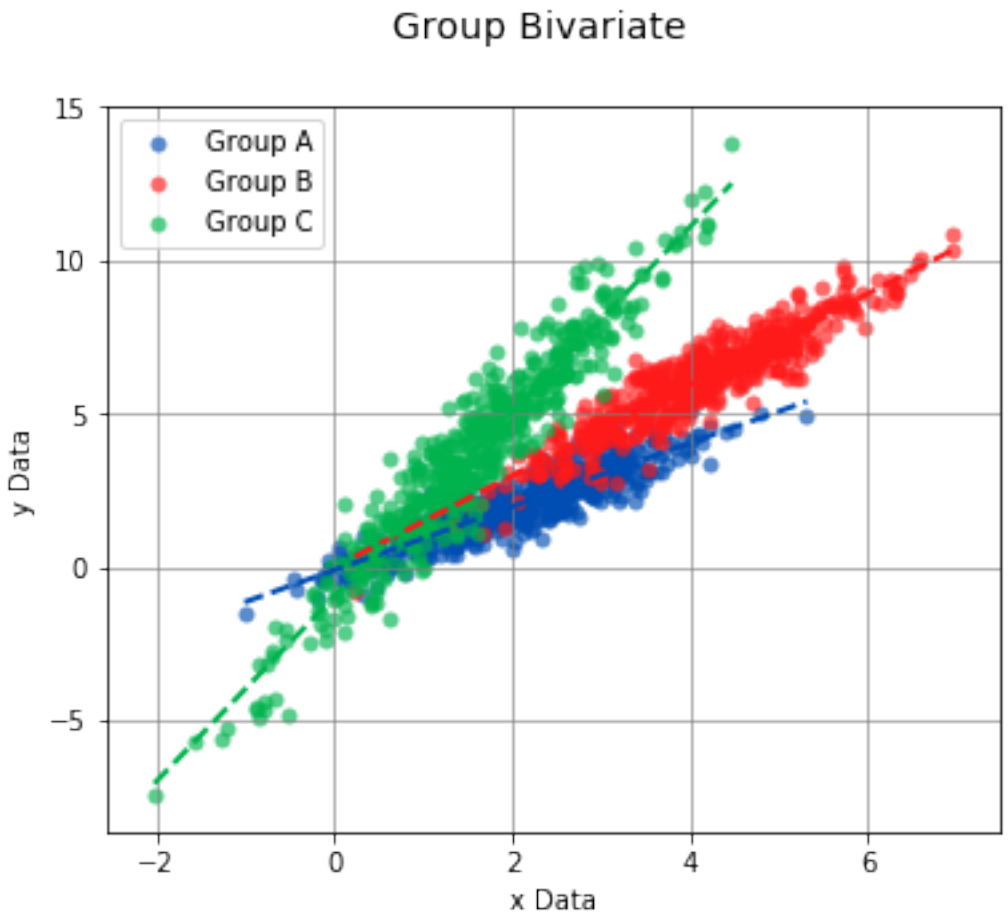
```
n         =  2000
Slope     =  0.0209
Intercept =  2.0554
r         =  0.0048
r^2       =  0.0000
Std Err   =  0.0964
p value   =  0.8285



Spearman Correlation Coefficient
--------------------------------

alpha   =  0.0500
r value =  0.0019
p value =  0.9320

H0: There is no significant relationship between predictor and response
```

Grouped Scatter Plot

If each set of data contains discrete and equivalent groups, the scatter plot can show each group in a separate color.

```
# Create new x-grouped and y-grouped from independent groups A, B, and C.
a_x = st.norm.rvs(2, size=500)
a_y = np.array([x + st.norm.rvs(0, 0.5, size=1) for x in a_x])

b_x = st.norm.rvs(4, size=500)
b_y = np.array([1.5 * x + st.norm.rvs(0, 0.65, size=1) for x in b_x])

c_x = st.norm.rvs(1.5, size=500)
c_y = np.array([3 * x + st.norm.rvs(0, 0.95, size=1) - 1 for x in c_x])

x_grouped = np.concatenate((a_x, b_x, c_x))
y_grouped = np.concatenate((a_y, b_y, c_y))

grps = np.concatenate((['Group A'] * 500, ['Group B'] * 500, ['Group C'] * 500))
```

```
analyze(
    x_grouped,
    y_grouped,
    groups=grps,
    boxplot_borders=False,
)
```



Linear Regression

n	Slope	Intercept	r^2	Std Err	p value	Group
500	1.0349	-0.0855	0.8042	0.0229	0.0000	Group A
500	1.4852	0.0162	0.8612	0.0267	0.0000	Group B
500	3.0115	-0.9385	0.9135	0.0415	0.0000	Group C

Pearson Correlation Coefficient

n	r value	p value	Group
500	0.8968	0.0000	Group A
500	0.9280	0.0000	Group B
500	0.9558	0.0000	Group C

Interpreting the Statistics

Linear Regression

The Linear Regression finds the least-squares best-fit line between the predictor and response. The linear relationship between the predictor and response is described by the relationship $y = mx + b$, where x is the predictor, y is the response, m is the slope, and b is the y-intercept.

- **n** - The number of data points in the analysis.
- **Slope** - The slope of the best-fit line between predictor and response.
- **Intercept** - The y-intercept of the best-fit line between predictor and response.
- **r** - The correlation coefficient of the linear regression.
- **r^2** - The amount of error that can be described by the linear regression. The higher the number, the more accurate the linear regression models the relationship between the predictor and response.
- **Std Err** - Standard error of the best-fit line.
- **p value** - The p value of the hypothesis test that the slope of the best-fit line is actually zero.

Correlation Coefficient

If the data points from both sets of data are normally distributed, the Pearson correlation coefficient is calculated, otherwise, the Spearman Rank correlation coefficient is calculated. A correlation coefficient of 0 indicates no relationship, whereas 1 indicates a perfect correlation between predictor and response. In the case of both correlation coefficients, the null hypothesis is that the correlation coefficient is 0, signifying no relationship between the predictor and response. If the p value is less than the significance $alpha$, the predictor and response are correlated.

Usage

analyze(*x-sequence*, *y-sequence*[, *fit=True*, *points=True*, *boxplot_borders=True*, *contours=False*, *labels=None*, *highlight=None*, *title='Bivariate'*, *xname='x Data'*, *yname='y Data'*, *save_to=None*])

Perform a Bivariate analysis on x-sequence and y-sequence.

Parameters

- **x-sequence** (*array-like*) – The array-like object on the x-axis (Predictor) to analyze. It can be a list, tuple, numpy array or pandas Series of numeric values.
- **y-sequence** (*array-like*) – The array-like object on the y-axis (Response) to analyze. It can be a list, tuple, numpy array or pandas Series of numeric values. The length of y-sequence should match the length of x-sequence.
- **groups** (*array-like*) – An array-like object of string values that correspond to groups to individually analyze. The length of groups should match the length of x-sequence and y-sequence.
- **fit** (*bool*) – Display the best fit line if True.
- **points** (*bool*) – Display the points on the scatter plot if True.
- **boxplot_borders** (*bool*) – Display boxplots along the x and y axis of the scatter plot if True.
- **contours** (*bool*) – Display the density contour lines on the scatter plot if True.

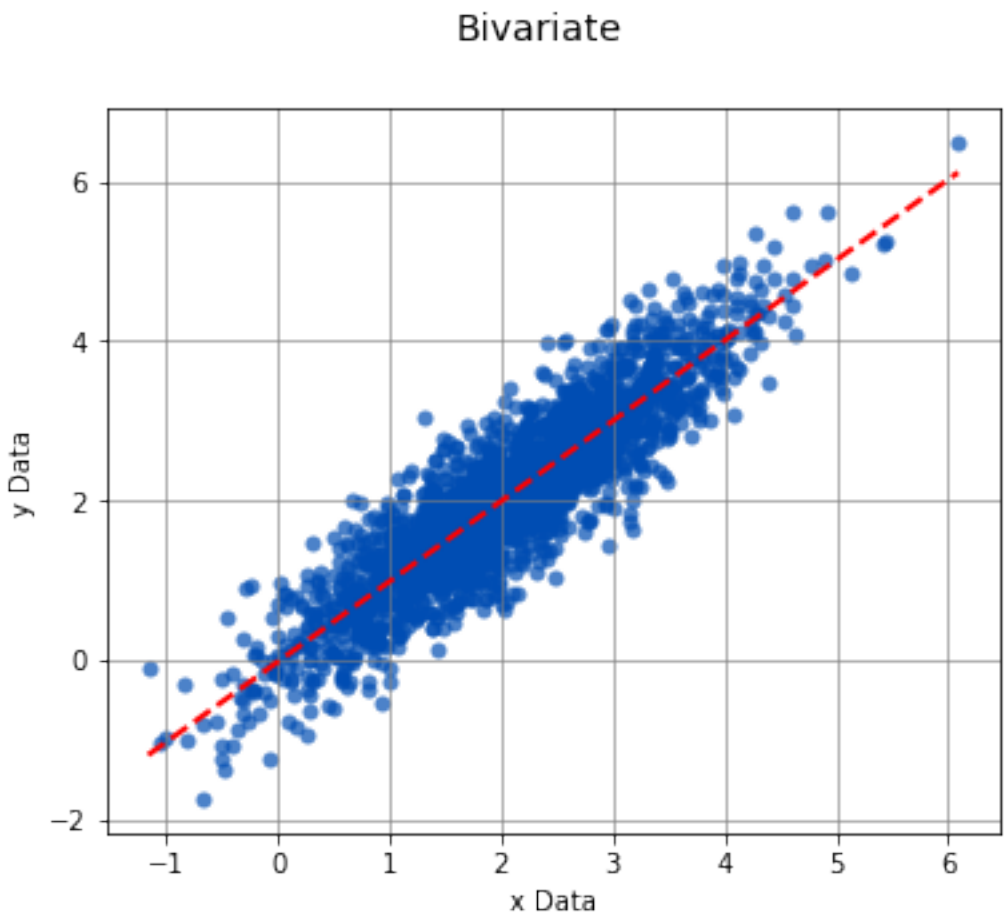
- **labels** (*array-like*) – Labels that identify the x and y data points in x-sequence and y-sequence. The length of labels should match the length of x-sequence and y-sequence.
- **highlight** (*array-like*) – A sequence of x and y data point labels to highlight on the scatter plot or a sequence of group names to highlight on the scatter plot. All other points will appear transparent on the scatter plot.
- **title** (*str*) – The title of the graph.
- **xname** (*str*) – The label of the x-axis of the scatter plot.
- **yname** (*str*) – The label of the y-axis of the scatter plot.

Argument Examples

x-sequence, y-sequence

The bare minimum requirements for performing a Bivariate analysis. The length of **x-sequence** and **y-sequence** should be equal and will raise an `UnequalVectorLengthError` if not.

```
analyze(
    x_sequence,
    y_sequence,
)
```

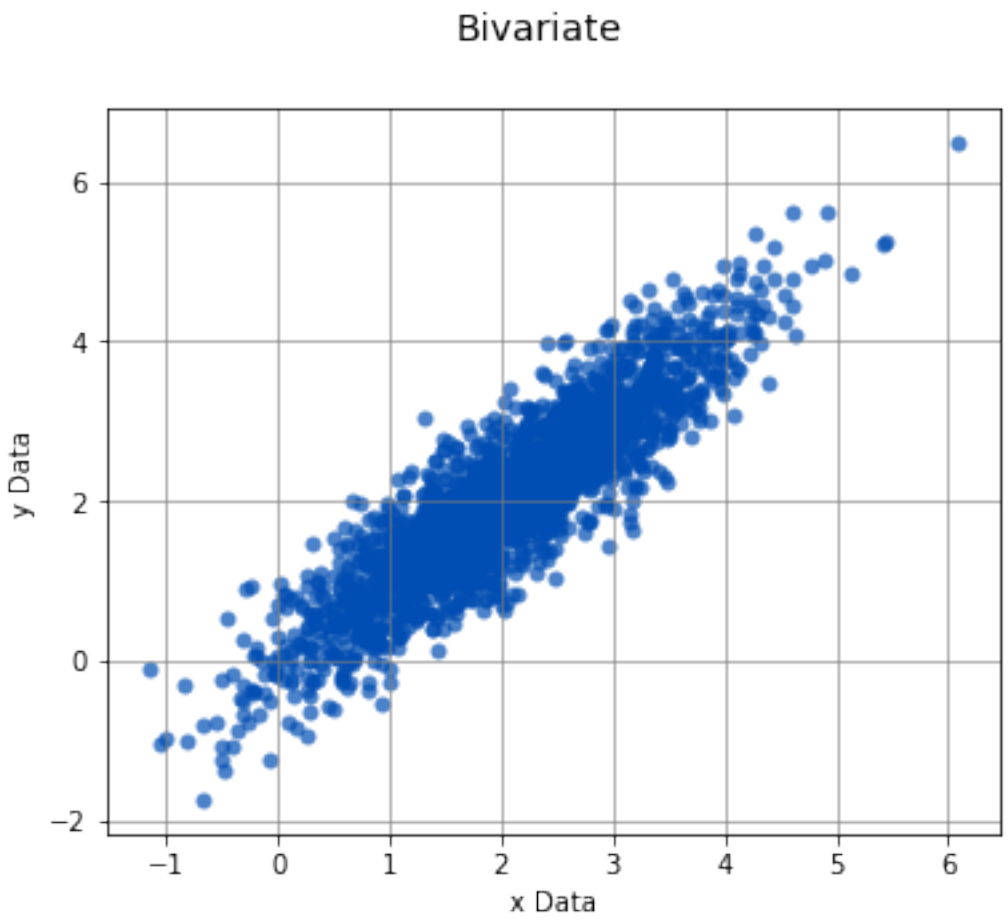


```
Linear Regression
-----------------

n         =  2000
Slope     =  1.0059
Intercept = -0.0133
r         =  0.8974
r^2       =  0.8054
Std Err   =  0.0111
p value   =  0.0000



Pearson Correlation Coefficient
-------------------------------

alpha   =  0.0500
r value =  0.8974
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

fit

Controls whether the best fit line is displayed or not.

```
analyze(
    x_sequence,
    y_sequence,
    fit=False,
)
```

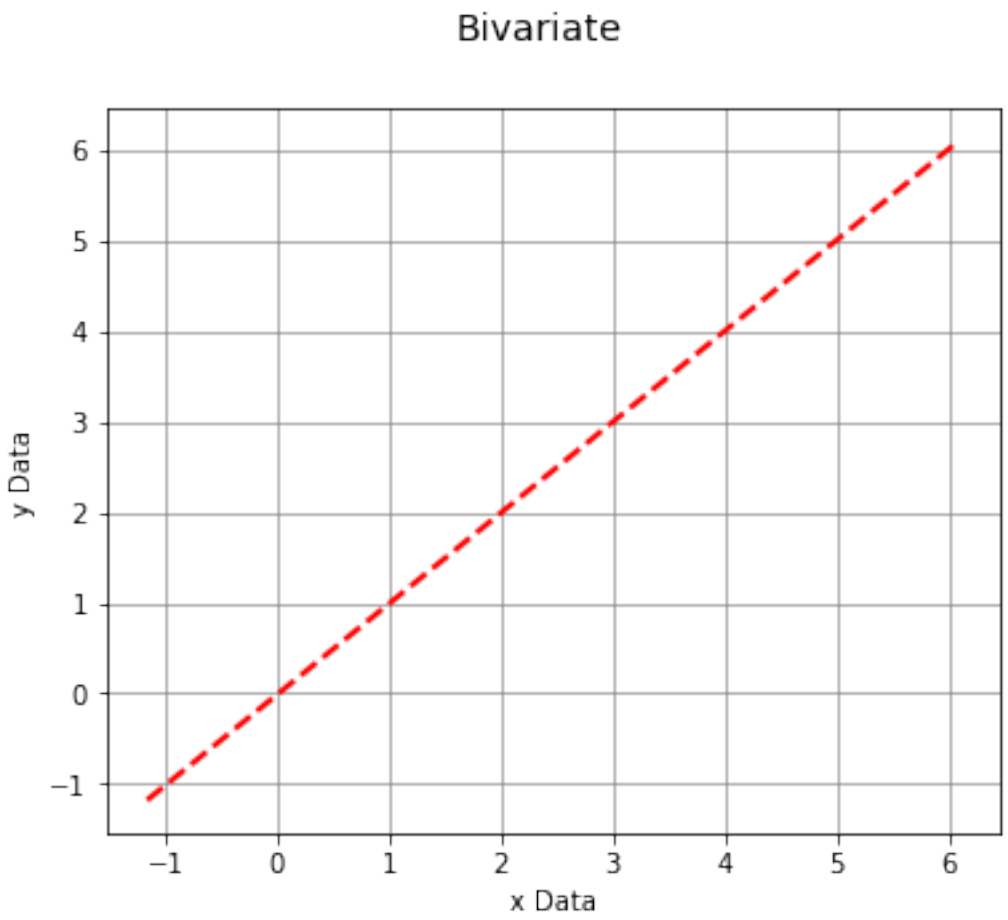



```
Linear Regression
-----------------

n         =  2000
Slope     =  1.0059
Intercept = -0.0133
r         =  0.8974
r^2       =  0.8054
Std Err   =  0.0111
p value   =  0.0000



Pearson Correlation Coefficient
-------------------------------

alpha   =  0.0500
r value =  0.8974
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

points

Controls whether the data points of the scatter plot are displayed or not.

```
analyze(
    x_sequence,
    y_sequence,
    points=False,
)
```




```
Linear Regression
-----------------

n         =  2000
Slope     =  1.0059
Intercept = -0.0133
r         =  0.8974
r^2       =  0.8054
Std Err   =  0.0111
p value   =  0.0000



Pearson Correlation Coefficient
```

(continues on next page)

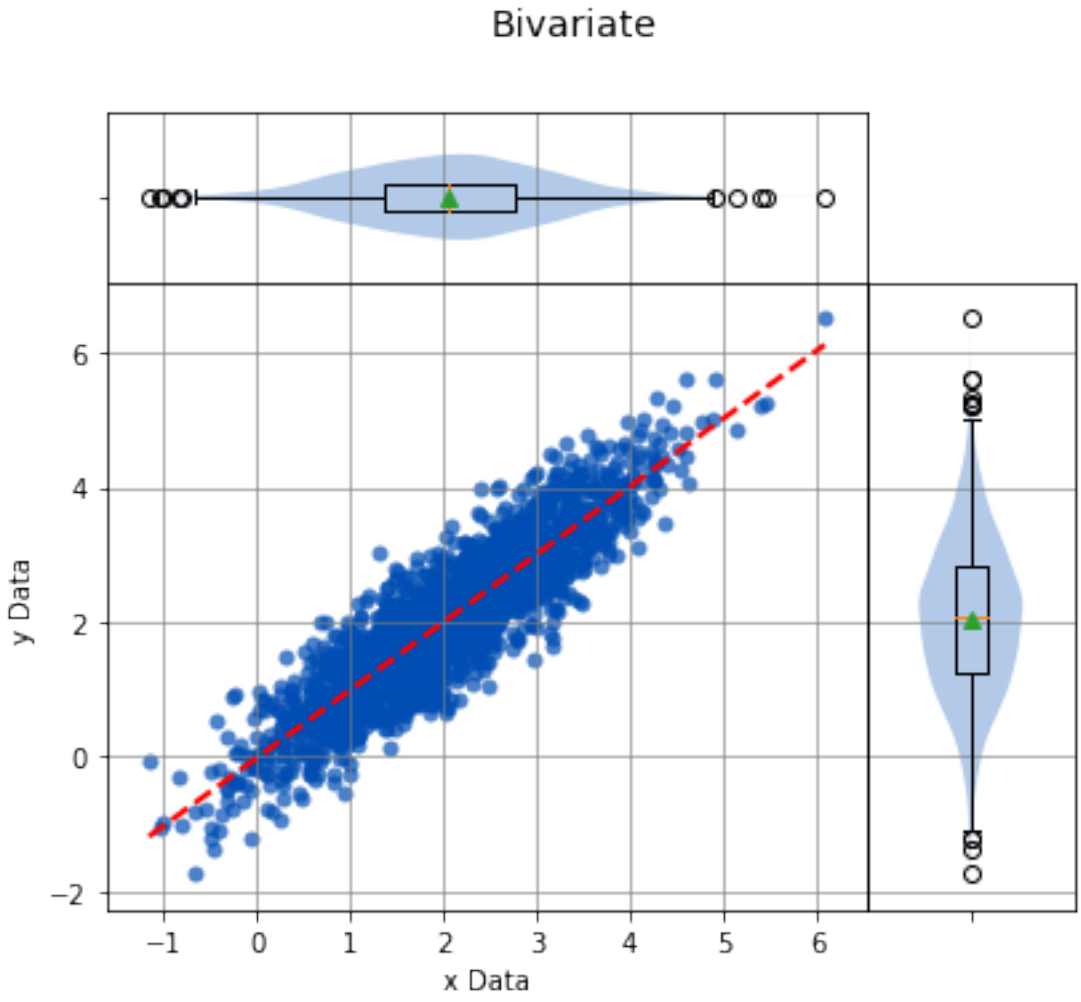
(continued from previous page)

```
------------------------------

alpha    =  0.0500
r value  =  0.8974
p value  =  0.0000

HA: There is a significant relationship between predictor and response
```

boxplot_borders

Controls whether boxplots are displayed for x-sequence and y-sequence.

```
analyze(
    x_sequence,
    y_sequence,
    boxplot_borders=True,
)
```

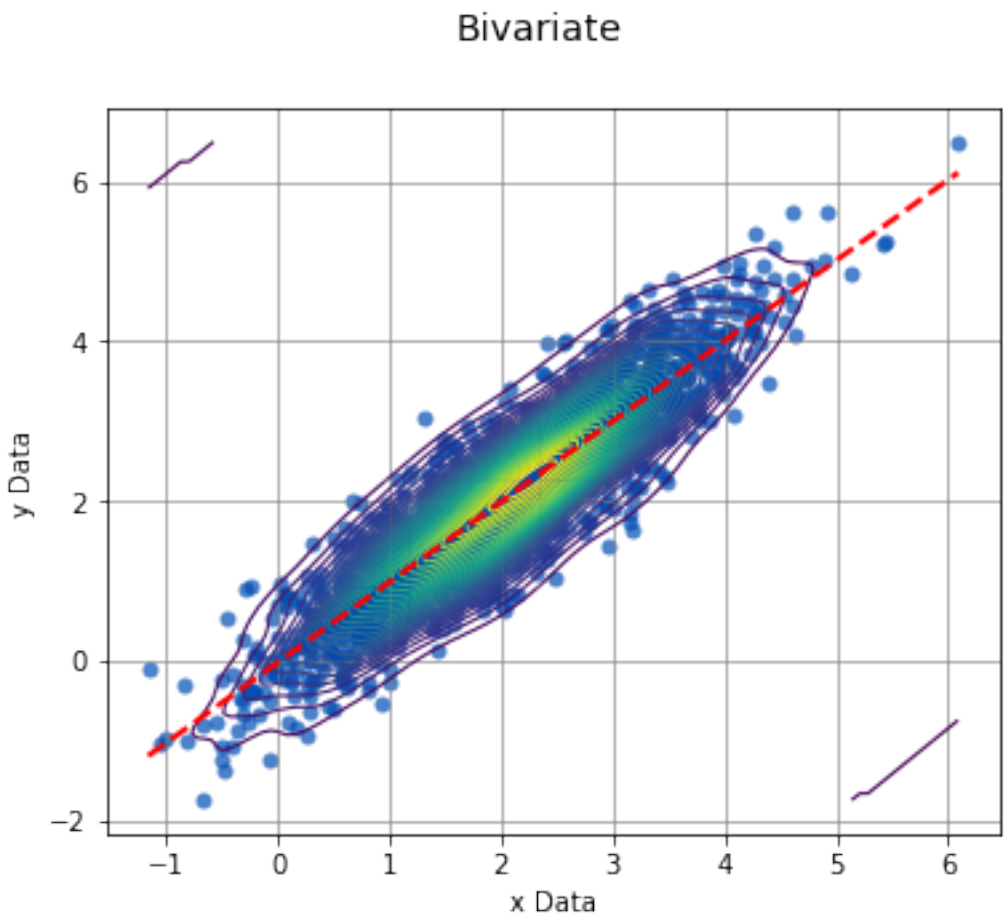


```
Linear Regression
-----------------

n         =  2000
Slope     =  1.0059
Intercept = -0.0133
r         =  0.8974
r^2       =  0.8054
Std Err   =  0.0111
p value   =  0.0000



Pearson Correlation Coefficient
-------------------------------

alpha   =  0.0500
r value =  0.8974
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

contours

Controls whether the density contours are displayed or not. The contours can be useful when analyzing joint probability distributions.

```
analyze(
    x_sequence,
    y_sequence,
    contours=True,
)
```



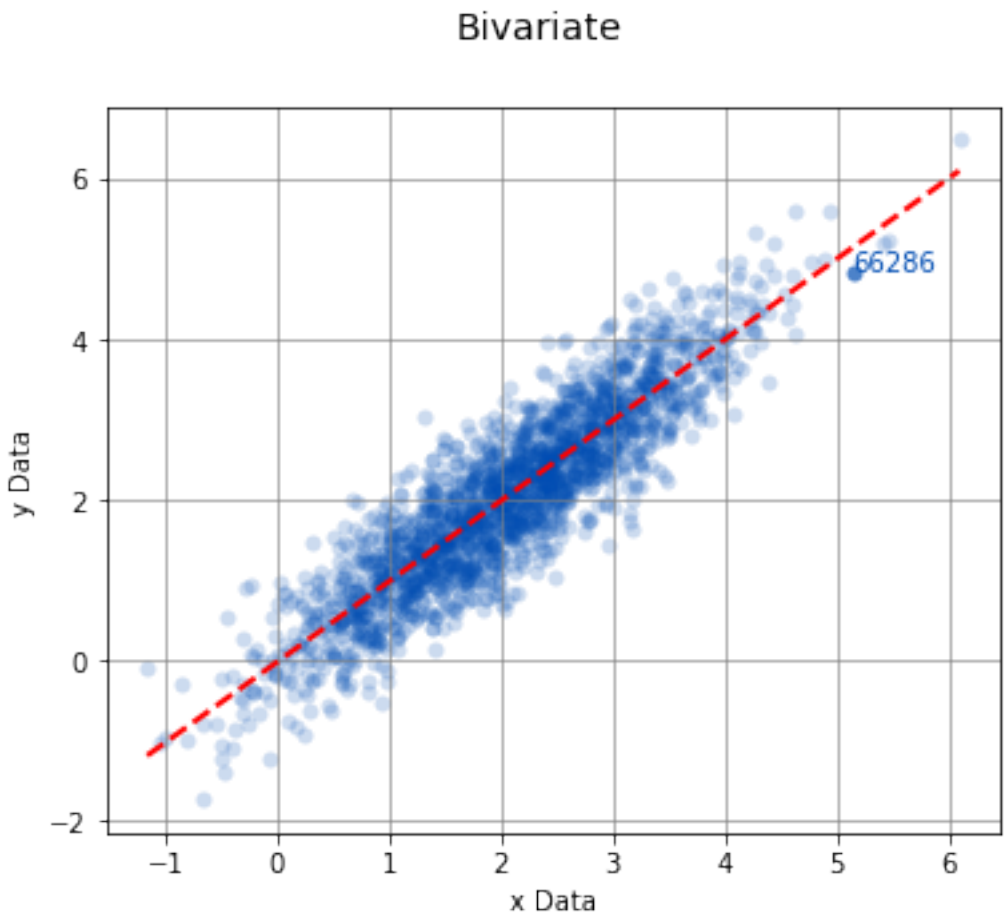

```
Linear Regression
-----------------

n         =  2000
Slope     =  1.0059
Intercept = -0.0133
r         =  0.8974
r^2       =  0.8054
Std Err   =  0.0111
p value   =  0.0000



Pearson Correlation Coefficient
-------------------------------

alpha   =  0.0500
r value =  0.8974
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

labels, highlight

Used in conjunction with one another, **labels** and **highlight** are used for displaying data values for the data points on the scatter plot.

```
labels = np.random.randint(low=10000, high=99999, size=2000)
```

```
analyze(
    x_sequence,
    y_sequence,
    labels=labels,
    highlight=[66286]
)
```




```
Linear Regression
-----------------

n         =  2000
Slope     =  1.0059
Intercept = -0.0133
r         =  0.8974
r^2       =  0.8054
Std Err   =  0.0111
p value   =  0.0000
```

(continues on next page)

(continued from previous page)

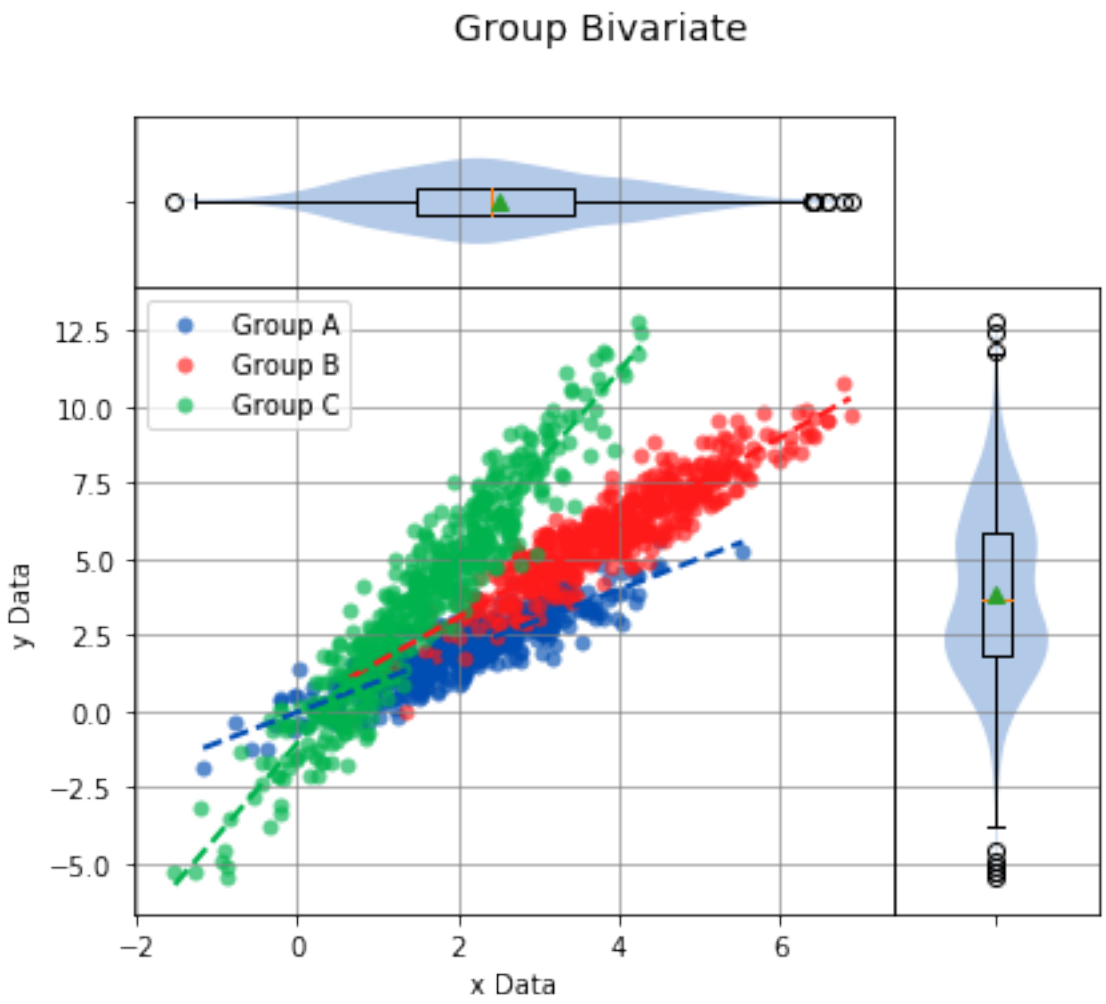
```
Pearson Correlation Coefficient
-------------------------------

alpha   =  0.0500
r value =  0.8974
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

groups

The **groups** argument can be used to perform a Bivariate analysis on separate collections of data points that can be compared to one another.

```
# Create new x-grouped and y-grouped from independent groups A, B, and C.
a_x = st.norm.rvs(2, size=500)
a_y = np.array([x + st.norm.rvs(0, 0.5, size=1) for x in a_x])

b_x = st.norm.rvs(4, size=500)
b_y = np.array([1.5 * x + st.norm.rvs(0, 0.65, size=1) for x in b_x])

c_x = st.norm.rvs(1.5, size=500)
c_y = np.array([3 * x + st.norm.rvs(0, 0.95, size=1) - 1 for x in c_x])

x_grouped = np.concatenate((a_x, b_x, c_x))
y_grouped = np.concatenate((a_y, b_y, c_y))

grps = np.concatenate((['Group A'] * 500, ['Group B'] * 500, ['Group C'] * 500))
```

```
analyze(
    x_grouped,
    y_grouped,
    groups=grps,
)
```




```
Linear Regression
-----------------

n             Slope         Intercept     r^2           Std Err       p value       
↪Group
-------------------------------------------------------------------------------------
↪------------
500            1.0109       -0.0389        0.7756        0.0244        0.0000       
↪Group A
500            1.4782        0.1239        0.8339        0.0296        0.0000       
↪Group B
500            3.0592       -1.0666        0.9084        0.0435        0.0000       
↪Group C


Pearson Correlation Coefficient
-------------------------------

n             r value       p value       Group
-----------------------------------------------------
500            0.8807        0.0000       Group A
500            0.9132        0.0000       Group B
```

(continues on next page)

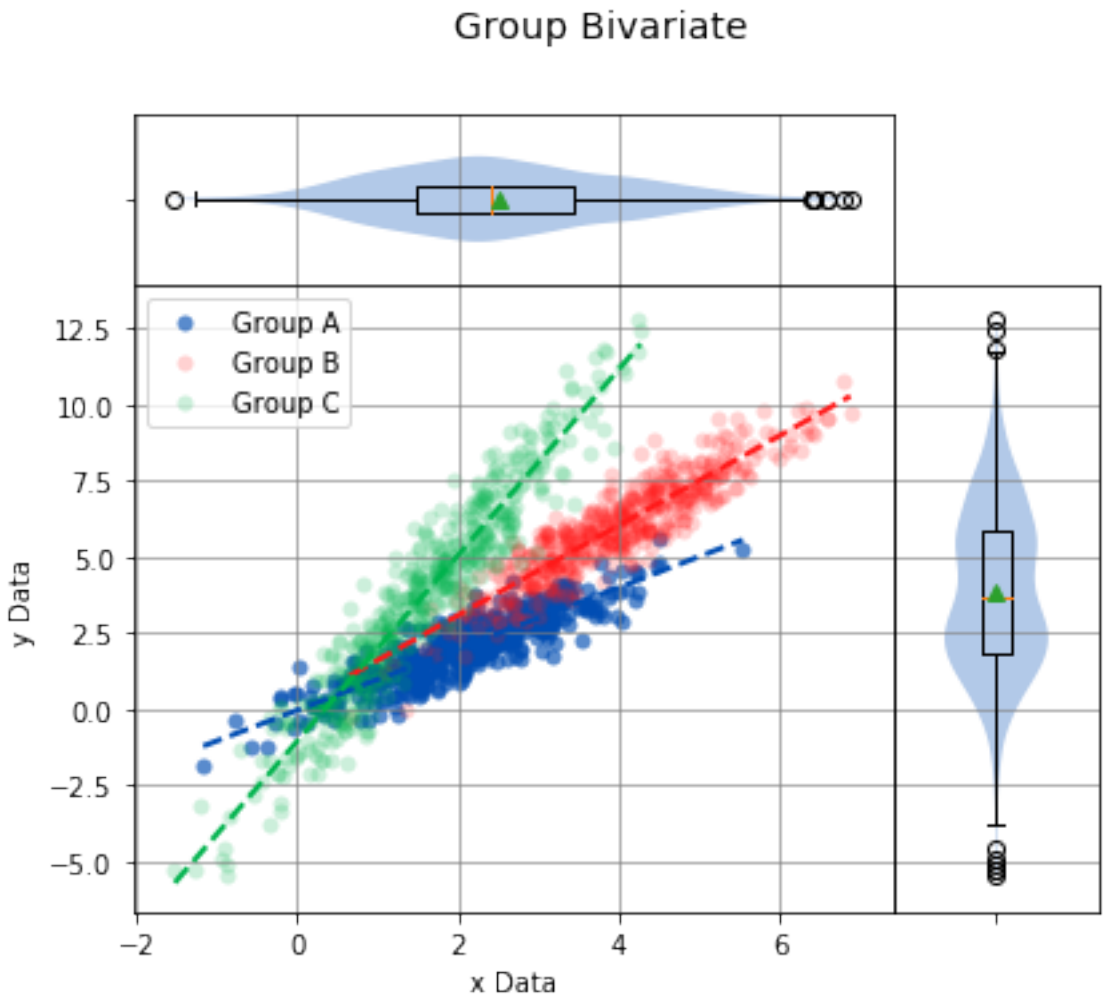
(continued from previous page)

```
500             0.9531          0.0000          Group C
```

groups, highlight

Using the **groups** argument is a great way to compare treatments. When combined with the **highlight** argument, a particular group can be highlighted on the scatter plot to stand out from the others.

```
analyze(
    x_grouped,
    y_grouped,
    groups=grps,
    highlight=['Group A'],
)
```




```
Linear Regression
-----------------

n               Slope           Intercept       r^2             Std Err         p value
↪Group
```

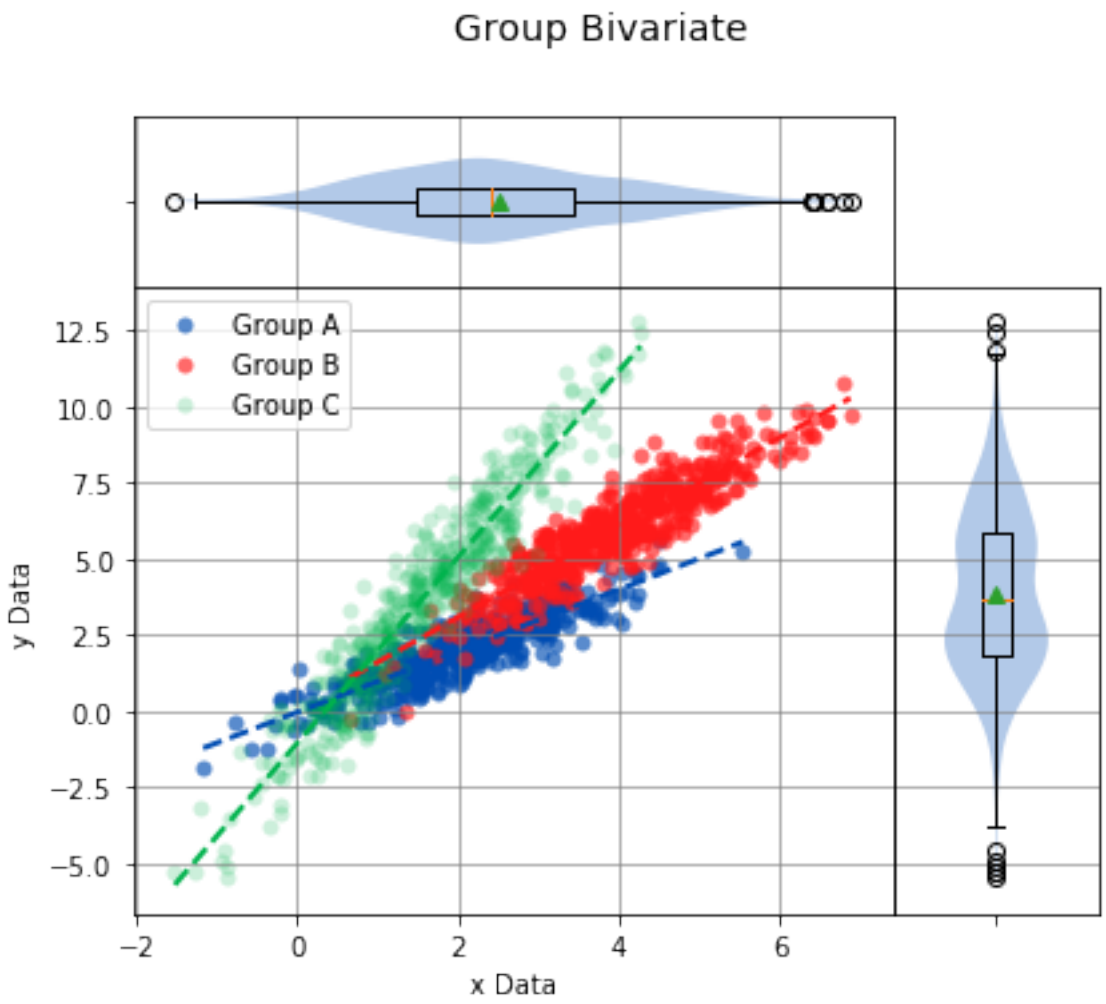
(continues on next page)

(continued from previous page)

```
-------------------------------------------------------------------------------------------
↪------------
500             1.0109       -0.0389        0.7756        0.0244        0.0000
↪Group A
500             1.4782        0.1239        0.8339        0.0296        0.0000
↪Group B
500             3.0592       -1.0666        0.9084        0.0435        0.0000
↪Group C


Pearson Correlation Coefficient
-------------------------------

n              r value       p value       Group
-----------------------------------------------------------
500             0.8807        0.0000       Group A
500             0.9132        0.0000       Group B
500             0.9531        0.0000       Group C
```

Multiple groups can also be highlighted.

```
analyze(
    x_grouped,
    y_grouped,
    groups=grps,
    highlight=['Group A', 'Group B'],
)
```



Linear Regression

n	Slope	Intercept	r^2	Std Err	p value	Group
500	1.0109	-0.0389	0.7756	0.0244	0.0000	Group A
500	1.4782	0.1239	0.8339	0.0296	0.0000	Group B
500	3.0592	-1.0666	0.9084	0.0435	0.0000	Group C

Pearson Correlation Coefficient

n	r value	p value	Group
500	0.8807	0.0000	Group A
500	0.9132	0.0000	Group B

(continues on next page)

(continued from previous page)

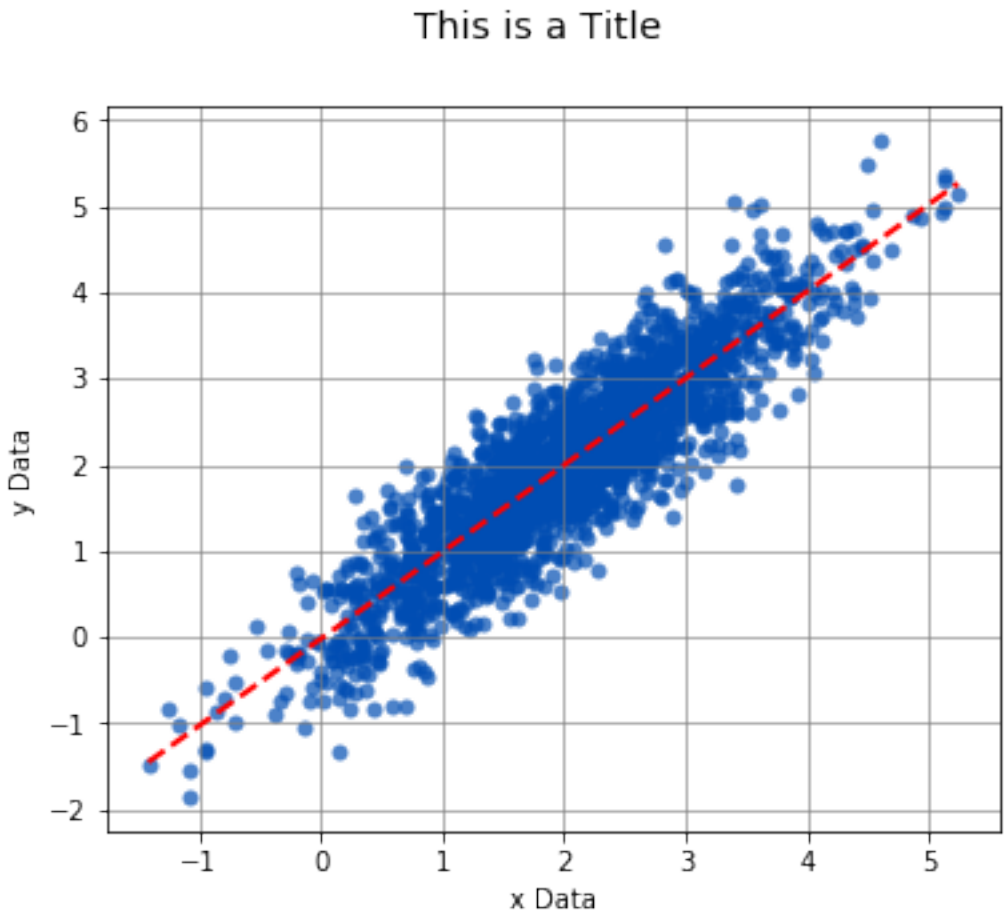
```
500             0.9531          0.0000          Group C
```

title

The title of the distribution to display above the graph.

```
x_sequence = st.norm.rvs(2, size=2000)
y_sequence = np.array([x + st.norm.rvs(0, 0.5, size=1) for x in x_sequence])
```

```
analyze(
    x_sequence,
    y_sequence,
    title='This is a Title',
)
```




```
Linear Regression
-----------------

n         =  2000
Slope     =  1.0086
Intercept = -0.0203
```

(continues on next page)

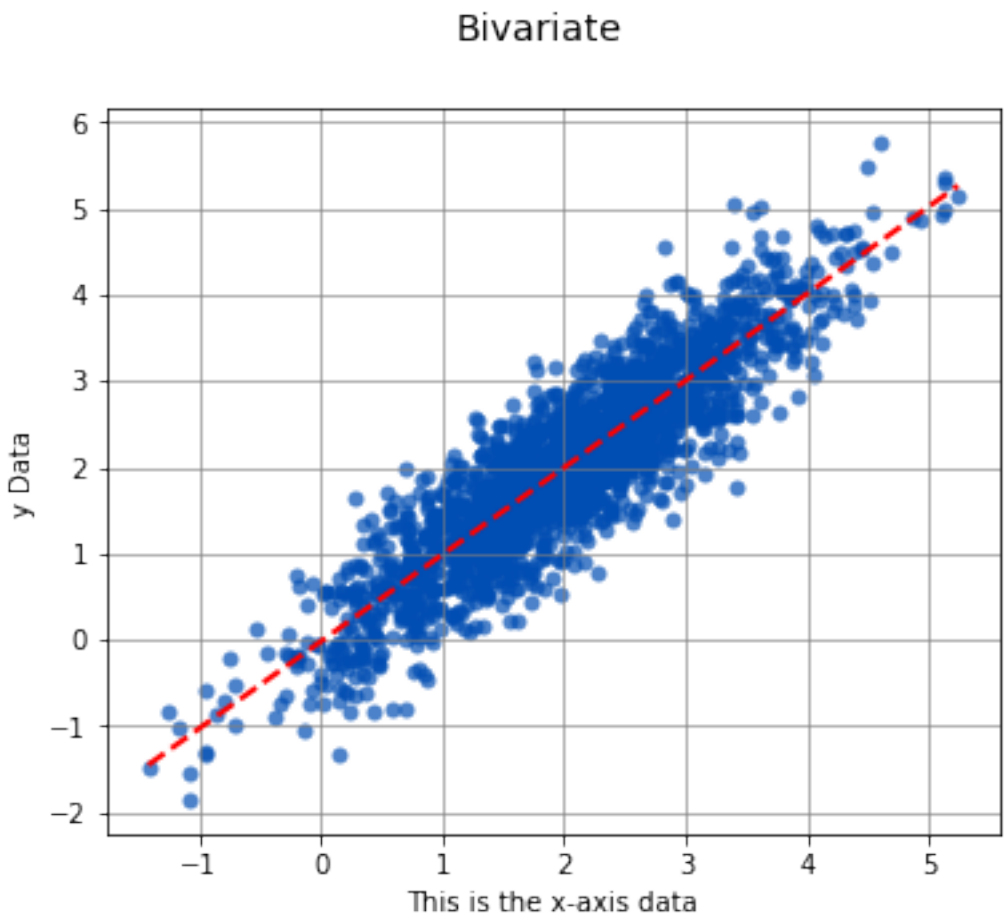
(continued from previous page)

```
r          =  0.8946
r^2        =  0.8004
Std Err    =  0.0113
p value    =  0.0000



Pearson Correlation Coefficient
-------------------------------

alpha   =  0.0500
r value =  0.8946
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

xname

The name of the data on the x-axis.

```
analyze(
    x_sequence,
    y_sequence,
    xname='This is the x-axis data'
)
```



Linear Regression

```
n         =  2000
Slope     =  1.0086
Intercept = -0.0203
r         =  0.8946
r^2       =  0.8004
Std Err   =  0.0113
p value   =  0.0000
```

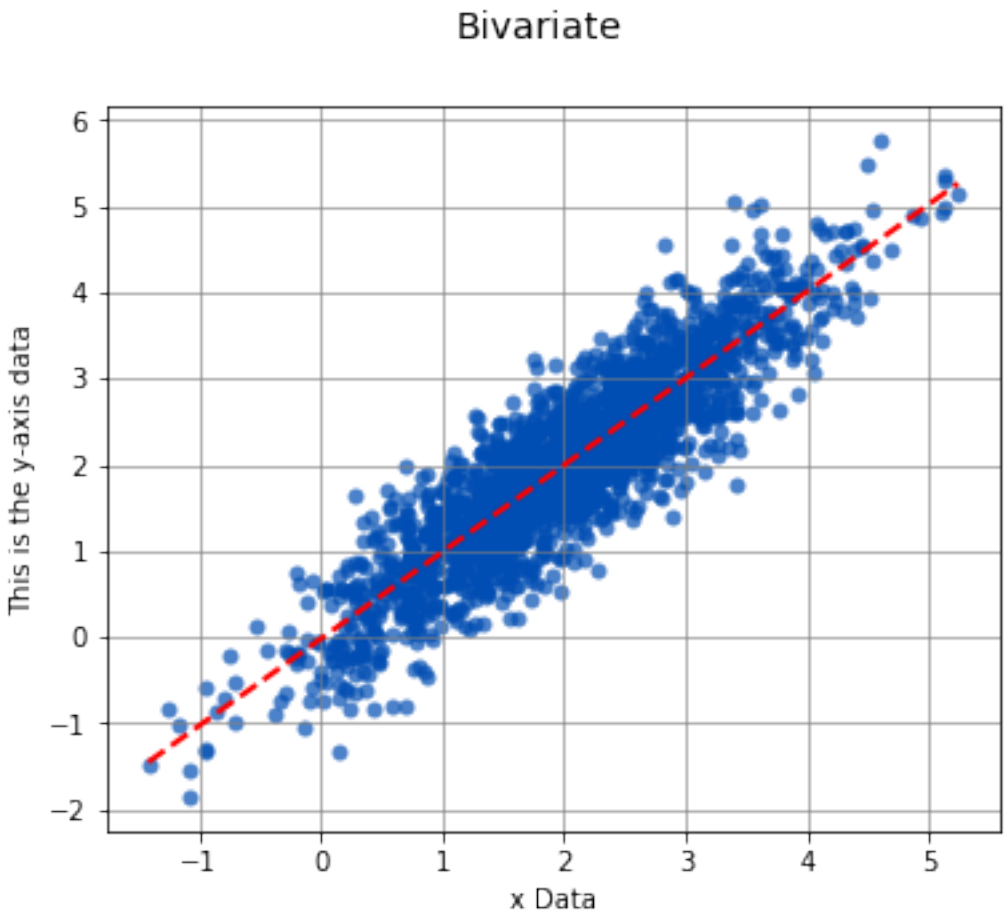
Pearson Correlation Coefficient

```
alpha   =  0.0500
r value =  0.8946
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

yname

The name of the data on the y-axis.

```
analyze(
    x_sequence,
    y_sequence,
    yname='This is the y-axis data'
)
```




```
Linear Regression
-----------------

n         =  2000
Slope     =  1.0086
Intercept = -0.0203
r         =  0.8946
r^2       =  0.8004
Std Err   =  0.0113
p value   =  0.0000



Pearson Correlation Coefficient
```

(continues on next page)

(continued from previous page)

```
------------------------------

alpha   =  0.0500
r value =  0.8946
p value =  0.0000

HA: There is a significant relationship between predictor and response
```

6.4.4 Location Test

Location testing is useful for comparing groups (also known as categories or treatments) of similar values to see if their locations are matched. In this case, location refers to a central value where all the values in a group have tendency to collect around. This is usually a mean or median of the group.

The Location Test analysis actually performs two tests, one for comparing variances between groups, and the second for comparing the location between groups. Both are useful for determining how similar or dissimilar the distribution of the groups are compared to one another.

Interpreting the Graphs

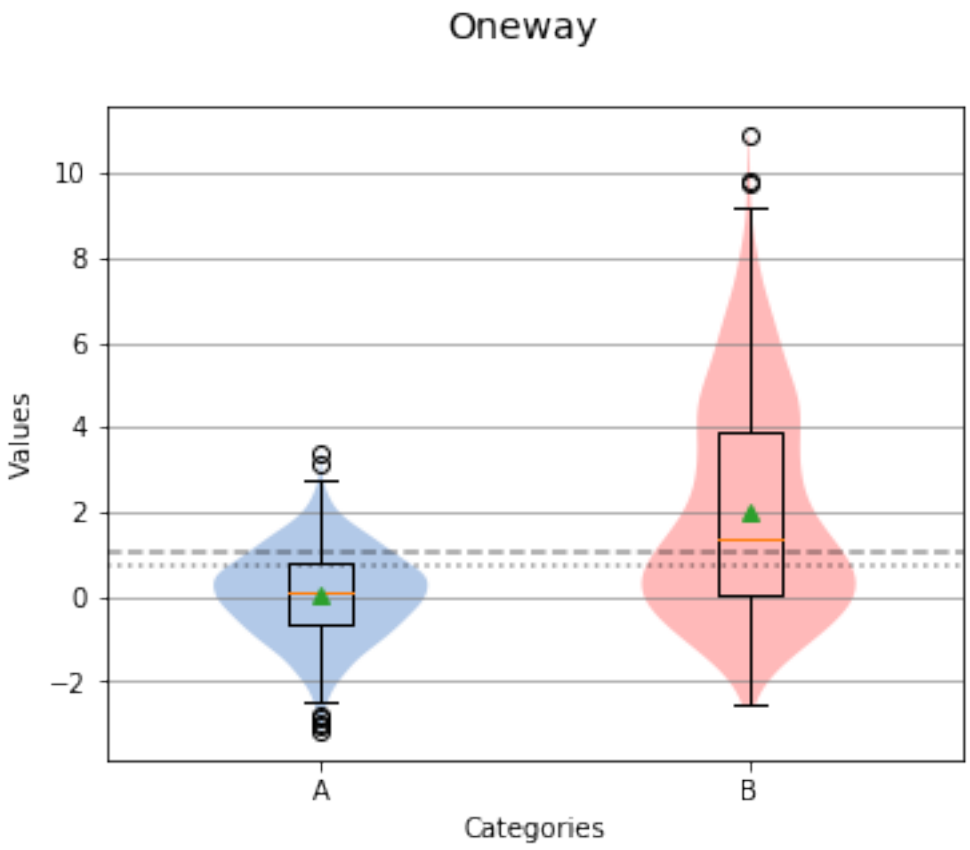
The graph produced by the Location Test produces three charts by default: Boxplots, Tukey-Kramer circles, and a Normal Quantile plot. Let's examine these individually.

```
import numpy as np
import scipy.stats as st
from sci_analysis import analyze

%matplotlib inline
```

The Boxplots

Boxplots in sci-analysis are actually a hybrid of two distribution visualization techniques, the boxplot and the violin plot. Boxplots are a good way to quickly understand a distribution, but can be misleading when the distribution is multimodal. A violin plot does a much better job at showing the local maxima and minima of a distribution.

```
np.random.seed(987654321)
a = st.norm.rvs(0, 1, 1000)
b = np.append(st.norm.rvs(4, 2, 500), st.norm.rvs(0, 1, 500))
analyze(
    {'A': a, 'B': b},
    circles=False,
    nqp=False,
)
```




```
Overall Statistics
------------------

Number of Groups =  2
Total            =  2000
Grand Mean       =  1.0413
Pooled Std Dev   =  1.9068
Grand Median     =  0.7279


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
------------------------------------------------------------------------------------
↪------------
1000          0.0551        1.0287        -3.1586       0.0897        3.4087        A
1000          2.0275        2.4926        -2.5414       1.3661        10.8915       B


Levene Test
-----------

alpha    =  0.0500
W value  =  513.4363
p value  =  0.0000
```

(continues on next page)

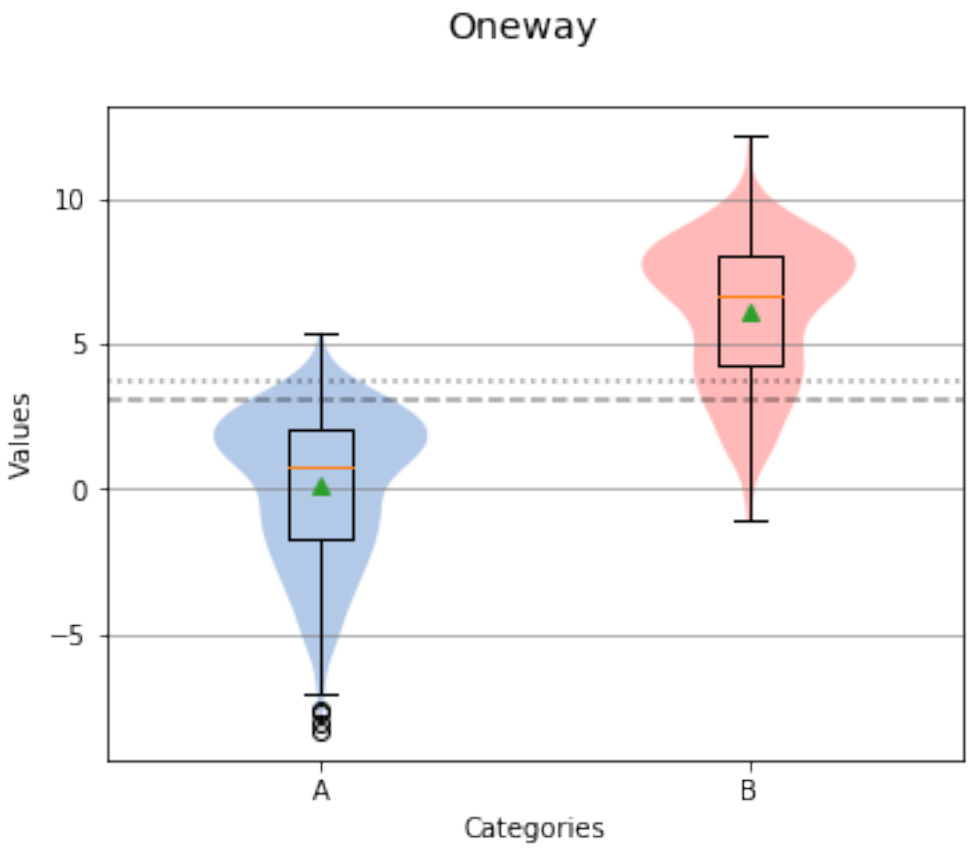
(continued from previous page)

```
HA: Variances are not equal



Mann Whitney U Test
-------------------

alpha   =  0.0500
u value =  263634.0000
p value =  0.0000

HA: Locations are not matched
```

In the center of each box is a red line and green triangle. The green triangle represents the mean of the group while the red line represents the median, sometimes referred to as the second quartile (Q2) or 50% line.

The boxplot graph also shows a short dotted line and long dotted line that represent the grand median and grand mean respectively.

```
np.random.seed(987654321)
a = np.append(st.norm.rvs(2, 1, 500), st.norm.rvs(-2, 2, 500))
b = np.append(st.norm.rvs(8, 1, 500), st.norm.rvs(4, 2, 500))
analyze(
    {'A': a, 'B': b},
    circles=False,
    nqp=False,
)
```




```
Overall Statistics
------------------

Number of Groups =  2
Total            =  2000
Grand Mean       =  3.0982
Pooled Std Dev   =  2.4841
Grand Median     =  3.7150


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
------------------------------------------------------------------------------------
↪------------
1000          0.0957        2.5307        -8.3172       0.7375        5.4087        A
1000          6.1006        2.4365        -1.0829       6.6925        12.1766       B


Levene Test
-----------

alpha    =  0.0500
W value  =  0.6238
p value  =  0.4297
```

(continues on next page)

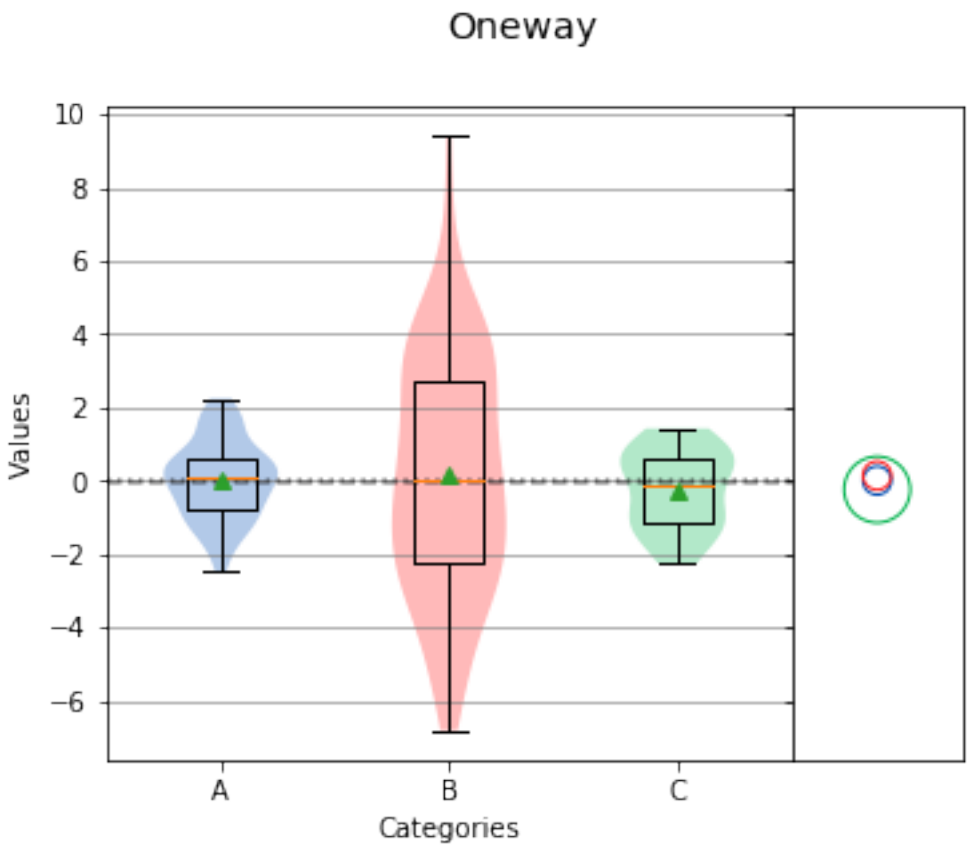
(continued from previous page)

```
H0: Variances are equal


Mann Whitney U Test
-------------------

alpha   =  0.0500
u value =  43916.0000
p value =  0.0000

HA: Locations are not matched
```

Tukey-Kramer Circles

Tukey-Kramer Circles, also referred to as comparison circles are based on the Tukey HSD test. Each circle is centered on the mean of each group and the radius of the circle is calculated from the mean standard error and size of the group. In this case, the radius is proportional to the standard error and inversely proportional to the size of the group. Therefore, a higher variation or smaller group size will produce a larger circle.

```
np.random.seed(987654321)
a = st.norm.rvs(0, 1, 100)
b = st.norm.rvs(0, 3, 100)
c = st.norm.rvs(0, 1, 20)
analyze(
    {'A': a, 'B': b, 'C': c},
    nqp=False,
)
```




```
Overall Statistics
------------------

Number of Groups =  3
Total            =  220
Grand Mean       = -0.0286
Pooled Std Dev   =  2.3353
Grand Median     =  0.0394


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
------------------------------------------------------------------------------------
↪------------
100            0.0083        1.0641       -2.4718        0.0761        2.2466       A
100            0.1431        3.2552       -6.8034        0.0394        9.4199       B
20            -0.2373        1.0830       -2.2349       -0.1229        1.4290       C


Bartlett Test
-------------

alpha    =  0.0500
T value =  117.7279
```

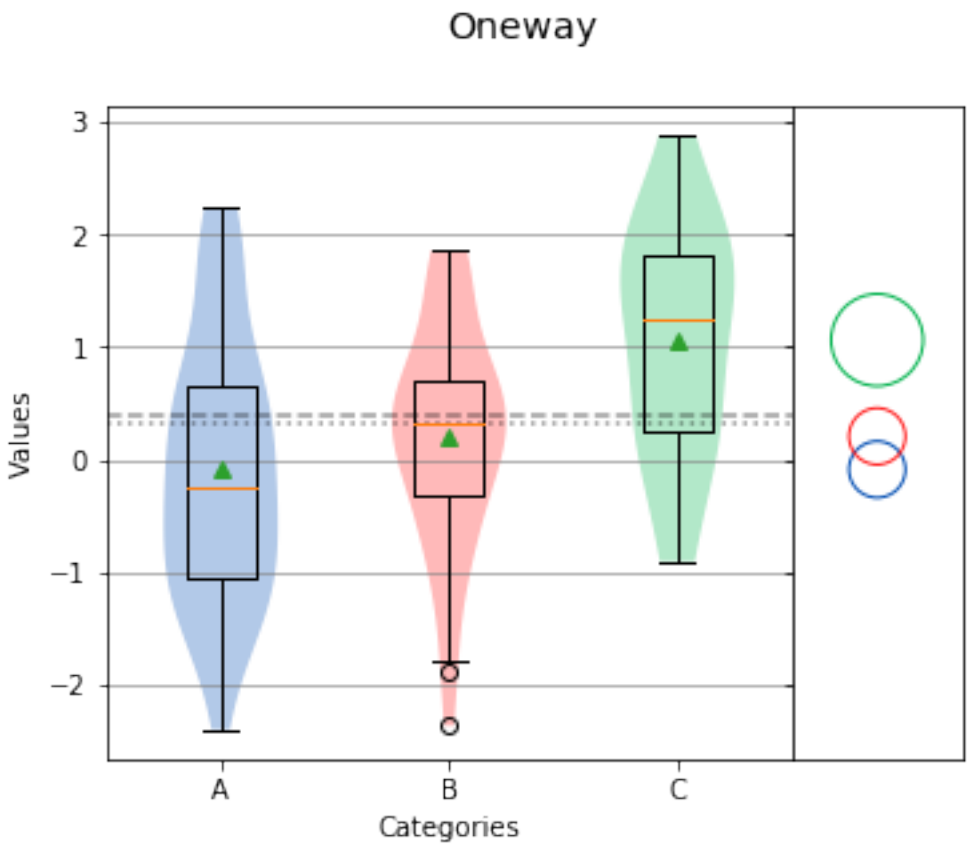
(continues on next page)

(continued from previous page)

```
p value =  0.0000

HA: Variances are not equal



Kruskal-Wallis
--------------

alpha   =  0.0500
h value =  0.2997
p value =  0.8608

H0: Group means are matched
```

If circles of different groups are mostly overlapping, the means of those groups are likely matched. However, if circles are not touching each other or only partly overlap, the means of those groups are likely different.

```
np.random.seed(987654321)
a = st.norm.rvs(0, 1, 50)
b = st.norm.rvs(0.1, 1, 50)
c = st.norm.rvs(1, 1, 20)
analyze(
    {'A': a, 'B': b, 'C': c},
    nqp=False,
)
```



```
Overall Statistics
------------------

Number of Groups =  3
Total            =  120
Grand Mean       =  0.3935
Pooled Std Dev   =  1.0608
Grand Median     =  0.3123


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           Group
------------------------------------------------------------------------------------------
50            -0.0891       1.1473        -2.4036       -0.2490       2.2466        A
50            0.2057        0.9758        -2.3718       0.3123        1.8617        B
20            1.0637        1.0391        -0.9072       1.2480        2.8849        C


Bartlett Test
-------------

alpha   =  0.0500
T value =  1.2811
p value =  0.5270

H0: Variances are equal



Oneway ANOVA
------------

alpha   =  0.0500
f value =  8.4504
p value =  0.0004

HA: Group means are not matched
```

Normal Quantile Plot

A Normal Quantile Plot is a specific type of Quantile-Quantile (Q-Q) plot where the quantiles on the x-axis correspond to the quantiles of the normal distribution. In the case of the Normal Quantile Plot, one quantile corresponds to one standard deviation.

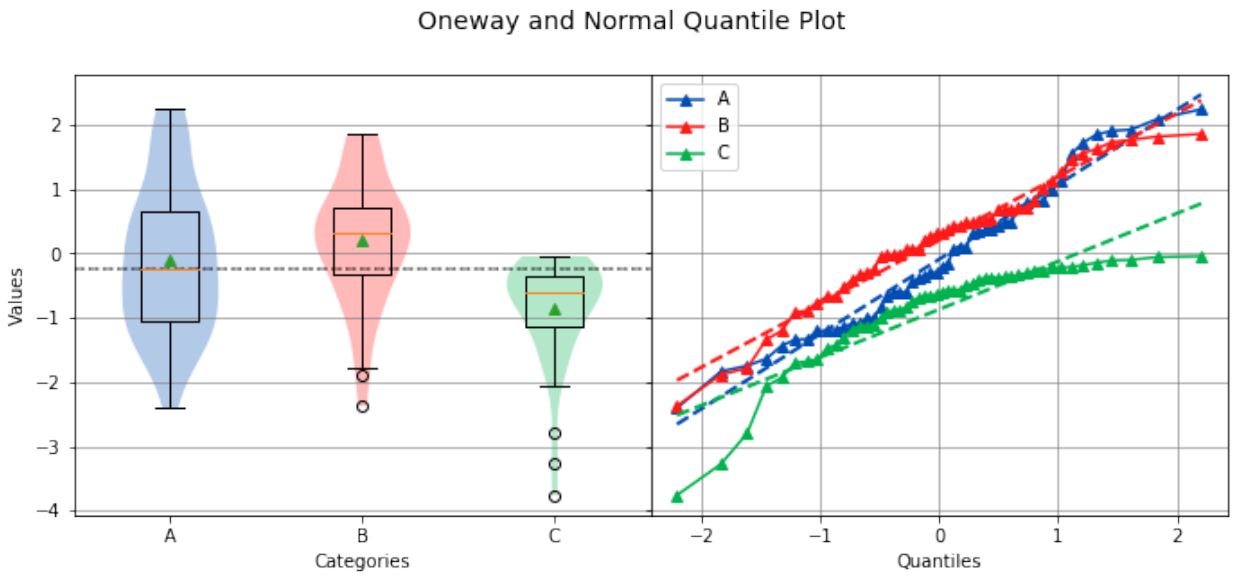
If the plotted points for a group on the Normal Quantile Plot closely resemble a straight line (regardless of slope), then the group is normally distributed. In the example below, group C is not normally distributed, as seen by it's downward curved shape on the Normal Quantile Plot.

```
np.random.seed(987654321)
a = st.norm.rvs(0, 1, size=50)
b = st.norm.rvs(0.1, 1, size=50)
c = st.weibull_max.rvs(0.95, size=50)
```

(continues on next page)

(continued from previous page)

```
analyze(
    {'A': a, 'B': b, 'C': c},
    circles=False,
)
```




```
Overall Statistics
------------------

Number of Groups =  3
Total            =  150
Grand Mean       = -0.2507
Pooled Std Dev   =  0.9868
Grand Median     = -0.2490


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
------------------------------------------------------------------------------------------
↪------------
50            -0.0891       1.1473        -2.4036       -0.2490       2.2466        A
50             0.2057       0.9758        -2.3718        0.3123       1.8617        B
50            -0.8687       0.8078        -3.7612       -0.6117       -0.0409       C


Levene Test
-----------

alpha    =  0.0500
W value  =  4.5142
p value  =  0.0125

HA: Variances are not equal
```

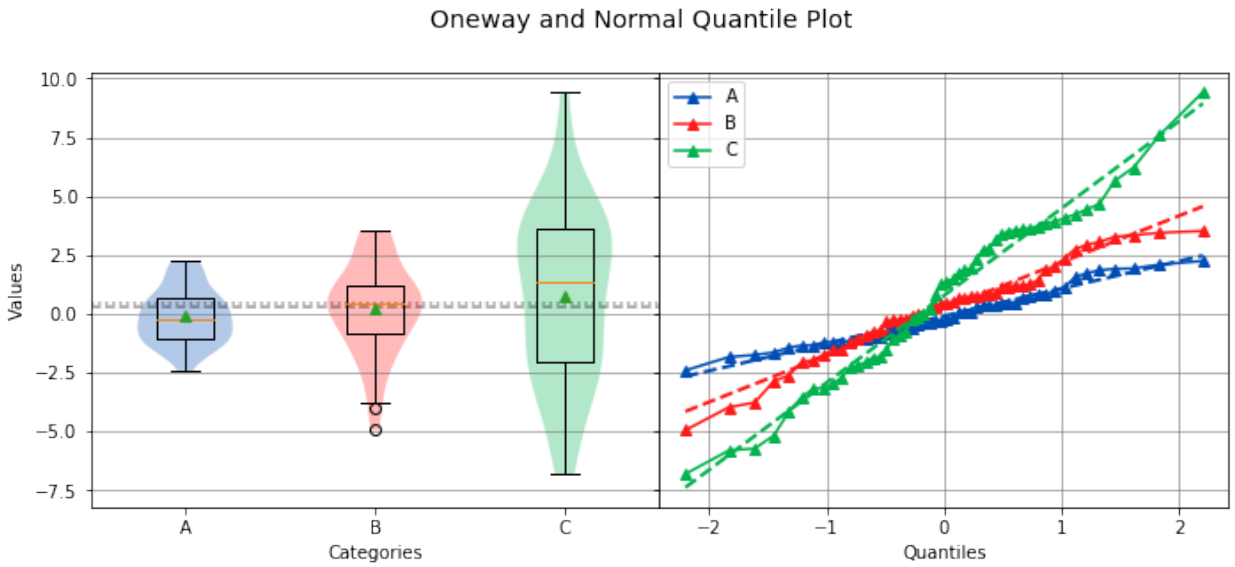
(continues on next page)

(continued from previous page)

```
Kruskal-Wallis
--------------

alpha    =  0.0500
h value  =  28.3558
p value  =  0.0000

HA: Group means are not matched
```

The slope of the data points on the Normal Quantile Plot indicate the relative variance of a particular group compared to the other groups.

```
np.random.seed(987654321)
a = st.norm.rvs(0, 1, 50)
b = st.norm.rvs(0, 2, 50)
c = st.norm.rvs(0, 3, 50)
analyze(
    {'A': a, 'B': b, 'C': c},
    circles=False,
)
```




```
Overall Statistics
------------------

Number of Groups =  3
Total            =  150
Grand Mean       =  0.3013
Pooled Std Dev   =  2.4717
Grand Median     =  0.4247


Group Statistics
----------------
```

(continues on next page)

(continued from previous page)

```
n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
-------------------------------------------------------------------------------------
↪------------
50            -0.0891       1.1473        -2.4036       -0.2490       2.2466        A
50             0.2113       1.9515        -4.9435        0.4247       3.5233        B
50             0.7816       3.6335        -6.8034        1.3194       9.4199        C


Bartlett Test
-------------

alpha   =  0.0500
T value =  60.0600
p value =  0.0000

HA: Variances are not equal



Kruskal-Wallis
--------------

alpha   =  0.0500
h value =  3.4335
p value =  0.1796

H0: Group means are matched
```

Interpreting the Statistics

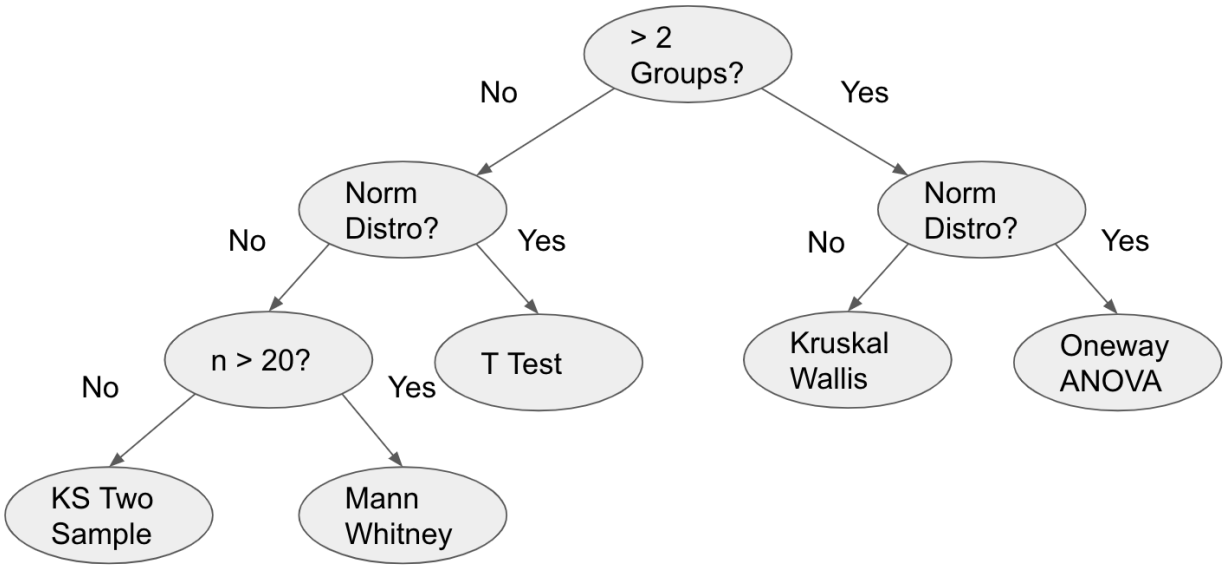
When performing a Location Test analysis, two statistics tables are given, the Overall Statistics and the Group Statistics.

The Overall Statistics shows the number of groups in the dataset, total number of data points in the dataset, Grand Mean, Grand Median, and Pooled Standard Deviation.

The Group Statistics list summary statistics for each group in a table. The summary statistics shown are the number of data points in the group (n), the Mean, Standard Deviation, Minimum, Median, Maximum, and group name.

The remaining two statistics are both Hypothesis Tests. The first test attempts to determine if the variances of each group are matched or not. The second test attempts to determine if the locations of each group are matched or not. Each hypothesis test shows the significance level (alpha), test statistic, and p-value. The hypothesis test used depends on a few different factors. The test for equal variance is fairly simple and depends on whether the all the data points in the dataset are normally distributed or not. If normally distributed, the Bartlett Test is used, otherwise the Levene Test is used.

The logic for determining which hypothesis test to use for checking location is more complex and depends on the number of groups, whether the data points in the dataset are normally distributed, and the size of the smallest group.



The five possible hypothesis tests from most sensitive to least sensitive are:

- Oneway ANOVA
- Kruskal Wallis
- Student's T-Test
- Mann Whitney
- Kolmogorov-Smirnov Two Sample Test

The last thing shown for each hypothesis test is the statement of the null hypothesis or alternative hypothesis. Each hypothesis has a null hypothesis that is assumed to be true. If the p-value of the test is lower than the significance level (alpha) of the test, the null hypothesis is rejected and the alternative hypothesis is stated. When the null hypothesis is rejected, it means that the likelihood of the outcome occurring by chance is significantly low enough that it is likely true.

Because the conclusion of hypothesis testing depends on an arbitrarily chosen significance level of 0.05, they should be taken with a bit of caution. This is why sci-analysis goes to lengths to try to use the most appropriate test given the supplied data and also pairs the test with graphs for a second source of truth.

Usage

Stacked Data

analyze(*sequence*, *groups*[, *nqp=True*, *circles=True*, *alpha=0.05*, *title='Oneway'*, *categories='Categories'*, *xname='Categories'*, *name='Values'*, *yname='Values'*, *save_to=None*])

Performs a location test of numeric, stacked data.

Parameters

- **sequence** (*array-like*) – The array-like object to analyze. It can be a list, tuple, numpy array or pandas Series of numeric values.
- **groups** (*array-like*) – An array-like of categorical values to group numeric values in *sequence* by. The values in groups correspond to the value at the same index in *sequence*. For this reason, the length of *sequence* and *groups* should be equal.

- **nqp** (*bool*) – Display the accompanying Normal Quantile Plot if **True**. The default value is **True**.
- **circles** (*bool*) – Display the Tukey-Kramer circles if **True**. The default value is **True**.
- **alpha** (*float*) – The significance level to use for hypothesis tests. The default value is 0.05.
- **title** (*str*) – The title of the graph.
- **categories** (*str*) – The label of the categories (groups) to be displayed along the x-axis of the graph.
- **xname** (*str*) – Alias for *categories*.
- **name** (*str*) – The label of the values in sequence to be displayed on the y-axis of the graph.
- **yname** (*str*) – Alias for *name*.
- **or None save_to** (*str*) – The path to the file where the graph will be saved.

Unstacked Data

analyze(*sequences*[, *groups=None*, *nqp=True*, *circles=True*, *alpha=0.05*, *title='Oneway'*, *categories='Categories'*, *xname='Categories'*, *name='Values'*, *yname='Values'*, *save_to=None*])

Performs a location test of numeric, unstacked data.

Parameters

- **or dict sequences** (*array-like*) – The object to analyze. If *sequences* is a dictionary, the keys will be used as the group names and the *groups* argument will be ignored. If *sequences* is an array-like, its values should be array-likes for each group to analyze. If *groups* is **None**, numbers will automatically be assigned as category names for each array-like in *sequences*.
- **groups** (*list*) – A list of categories to group values in *sequences* by. The order of values in *groups* should match the array-like values in *sequences*.
- **nqp** (*bool*) – Display the accompanying Normal Quantile Plot if **True**. The default value is **True**.
- **circles** (*bool*) – Display the Tukey-Kramer circles if **True**. The default value is **True**.
- **alpha** (*float*) – The significance level to use for hypothesis tests. The default value is 0.05.
- **title** (*str*) – The title of the graph.
- **categories** (*str*) – The label of the categories (groups) to be displayed along the x-axis of the graph.
- **xname** (*str*) – Alias for categories.
- **name** (*str*) – The label of the values in sequence to be displayed on the y-axis of the graph.
- **yname** (*str*) – Alias for name.
- **or None save_to** (*str*) – The path to the file where the graph will be saved.

Argument Examples

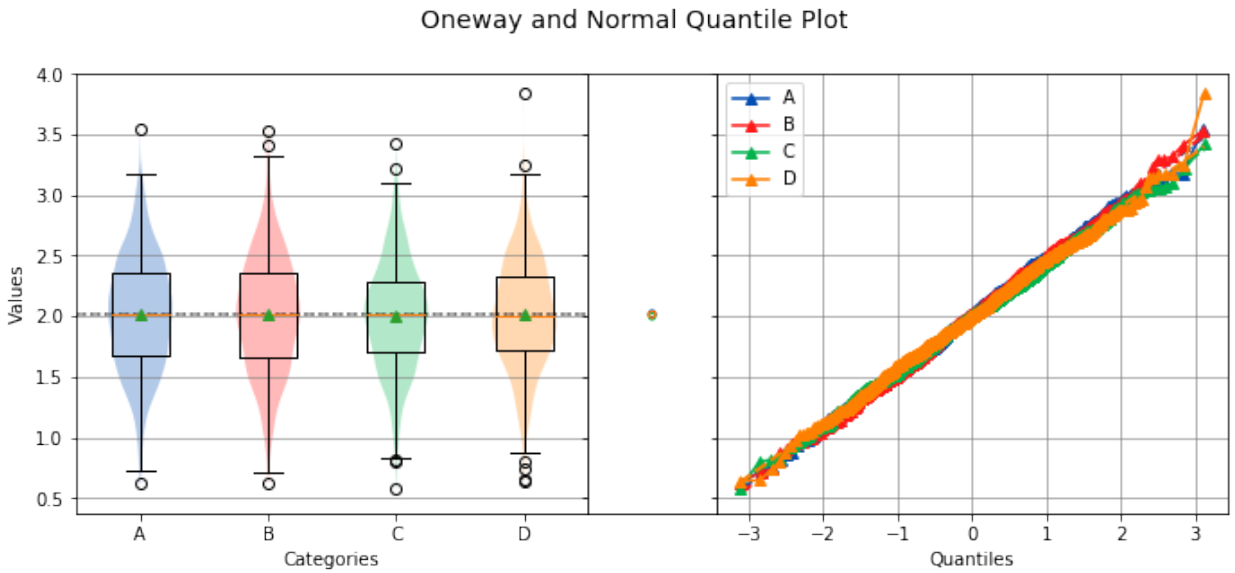
Let's first import sci-analysis and setup some variables to use in these examples.

```
# Create sequence and groups from random variables for stacked data examples.
stacked = st.norm.rvs(2, 0.45, size=3000)
vals = 'ABCD'
stacked_groups = []
for _ in range(3000):
    stacked_groups.append(vals[np.random.randint(0, 4)])
```

sequence, groups

When analyzing stacked data, both *sequence* and *groups* are required.

```
analyze(
    stacked,
    groups=stacked_groups,
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0118
Pooled Std Dev   =  0.4558
Grand Median     =  2.0174


Group Statistics
----------------

n              Mean           Std Dev        Min            Median         Max
↪Group
```

(continues on next page)

(continued from previous page)

```
-------------------------------------------------------------------------------
↪------------
720             2.0211        0.4667        0.6218        2.0240        3.5506        A
736             2.0126        0.4729        0.6281        2.0228        3.5339        B
782             1.9991        0.4403        0.5786        2.0120        3.4248        C
762             2.0143        0.4439        0.6396        2.0046        3.8397        D


Bartlett Test
-------------

alpha   =  0.0500
T value =  5.7422
p value =  0.1248

H0: Variances are equal



Oneway ANOVA
------------

alpha   =  0.0500
f value =  0.3117
p value =  0.8169

H0: Group means are matched
```

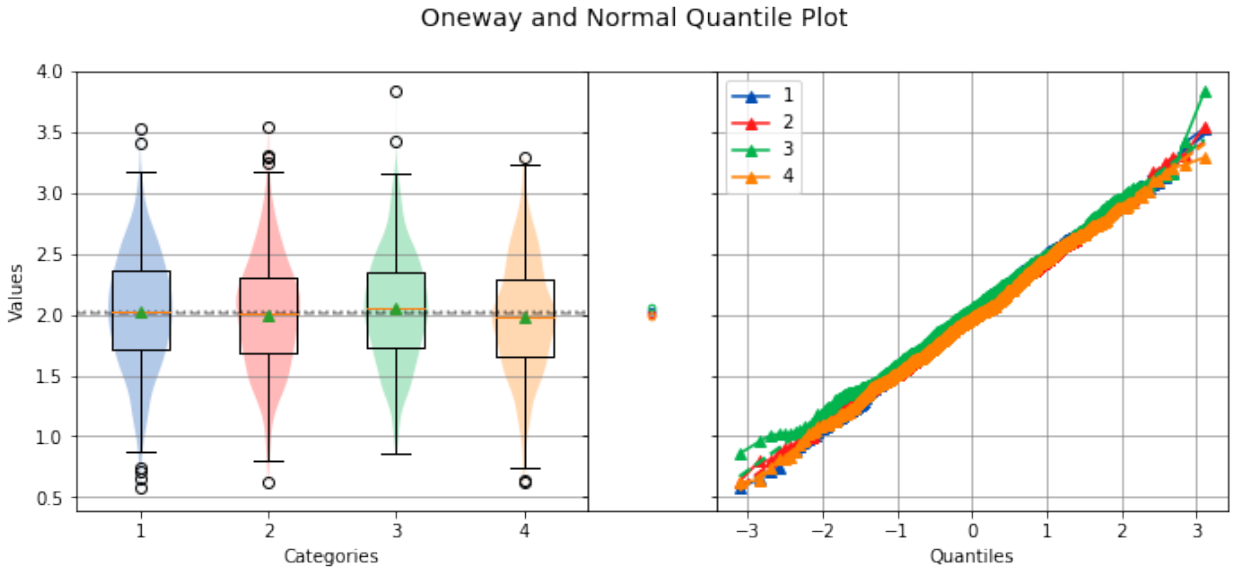
sequences

When analyzing unstacked data, *sequences* can be a dictionary or an array-like of array-likes.

```
# Create sequences from random variables for unstacked data examples.
np.random.seed(987654321)
a = st.norm.rvs(2, 0.45, size=750)
b = st.norm.rvs(2, 0.45, size=750)
c = st.norm.rvs(2, 0.45, size=750)
d = st.norm.rvs(2, 0.45, size=750)
```

If *sequences* is an array-like of array-likes, and *groups* is **None**, category labels will be automatically generated starting at 1.

```
analyze([a, b, c, d])
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0149
Pooled Std Dev   =  0.4564
Grand Median     =  2.0219


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
------------------------------------------------------------------------------------
↪------------
750           2.0234        0.4679        0.5786        2.0328        3.5339        1
750           2.0006        0.4553        0.6281        2.0110        3.5506        2
750           2.0538        0.4446        0.8564        2.0512        3.8397        3
750           1.9819        0.4575        0.6218        1.9780        3.2952        4


Bartlett Test
-------------

alpha    =  0.0500
T value  =  1.9719
p value  =  0.5783

H0: Variances are equal



Oneway ANOVA
------------
```

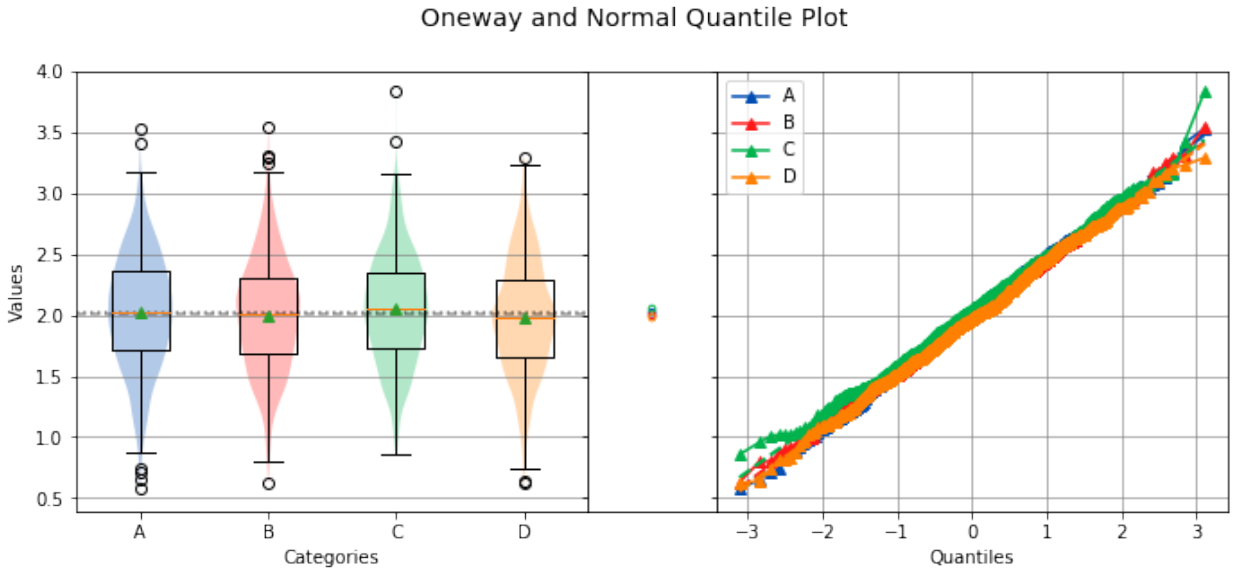
(continues on next page)

(continued from previous page)

```
alpha    =  0.0500
f value  =  3.4508
p value  =  0.0159

HA: Group means are not matched
```

If *sequences* is a dictionary, the keys will be used as category labels. .. note:: When *sequences* is a dictionary, the categories will not necessarily be shown in order.

```
analyze({'A': a, 'B': b, 'C': c, 'D': d})
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0149
Pooled Std Dev   =  0.4564
Grand Median     =  2.0219


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
------------------------------------------------------------------------------------------
↪------------
750           2.0234        0.4679        0.5786        2.0328        3.5339        A
750           2.0006        0.4553        0.6281        2.0110        3.5506        B
750           2.0538        0.4446        0.8564        2.0512        3.8397        C
750           1.9819        0.4575        0.6218        1.9780        3.2952        D


Bartlett Test
```

(continues on next page)

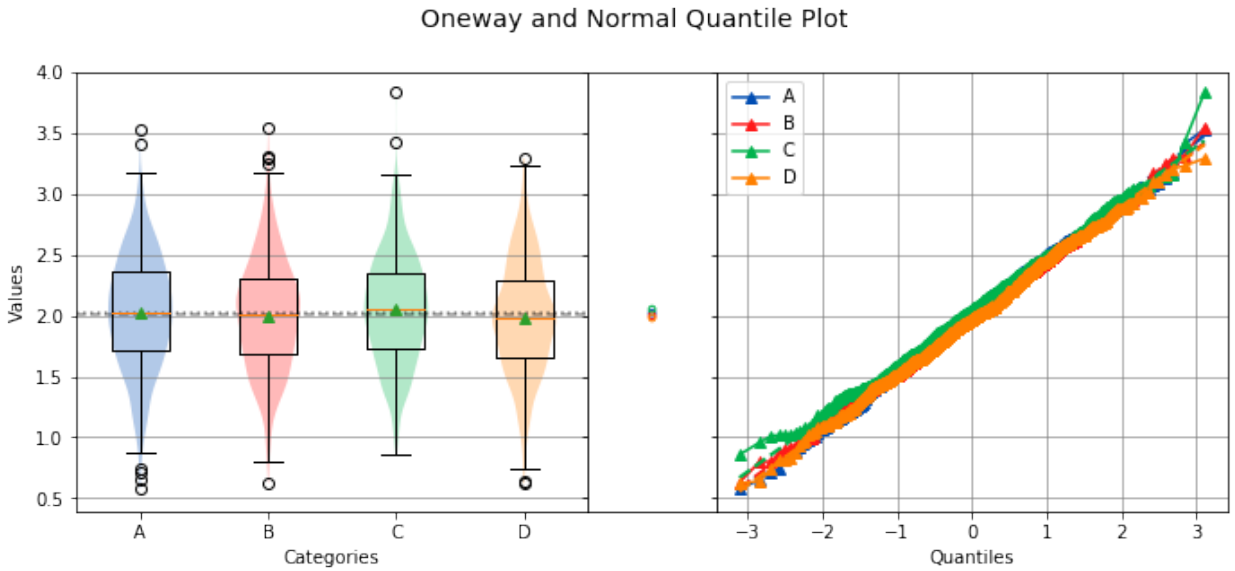
(continued from previous page)

```
-------------

alpha   =  0.0500
T value =  1.9719
p value =  0.5783

H0: Variances are equal


Oneway ANOVA
------------

alpha   =  0.0500
f value =  3.4508
p value =  0.0159

HA: Group means are not matched
```

groups

If analyzing stacked data, *groups* should be an array-like with the same length as *sequence*. If analyzing unstacked data, *groups* should be the same length as *sequences* and all values in l*groups* should be unique.

```
analyze(
    [a, b, c, d],
    groups=['A', 'B', 'C', 'D'],
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
```

(continues on next page)

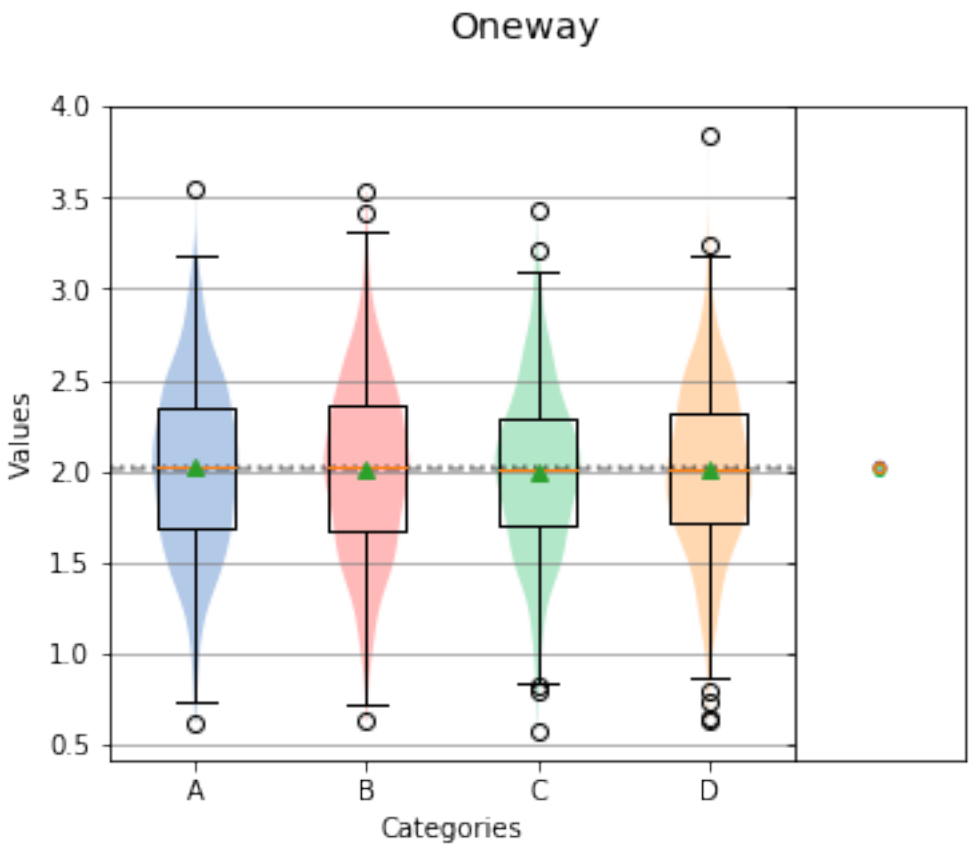
(continued from previous page)

```
Grand Mean       =  2.0149
Pooled Std Dev   =  0.4564
Grand Median     =  2.0219


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
-------------------------------------------------------------------------------------
↪------------
750           2.0234        0.4679        0.5786        2.0328        3.5339        A
750           2.0006        0.4553        0.6281        2.0110        3.5506        B
750           2.0538        0.4446        0.8564        2.0512        3.8397        C
750           1.9819        0.4575        0.6218        1.9780        3.2952        D


Bartlett Test
-------------

alpha   =  0.0500
T value =  1.9719
p value =  0.5783

H0: Variances are equal



Oneway ANOVA
------------

alpha   =  0.0500
f value =  3.4508
p value =  0.0159

HA: Group means are not matched
```

nqp

Controls whether the Normal Quantile Plot is displayed or not. The default value is **True**.

```
analyze(
    stacked,
    groups=stacked_groups,
    nqp=False,
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0118
Pooled Std Dev   =  0.4558
Grand Median     =  2.0174


Group Statistics
----------------

n              Mean           Std Dev        Min            Median         Max            ␣
↪Group
-------------------------------------------------------------------------------------------
↪------------
720            2.0211         0.4667         0.6218         2.0240         3.5506         A
736            2.0126         0.4729         0.6281         2.0228         3.5339         B
782            1.9991         0.4403         0.5786         2.0120         3.4248         C
762            2.0143         0.4439         0.6396         2.0046         3.8397         D


Bartlett Test
-------------

alpha    =  0.0500
```

(continues on next page)

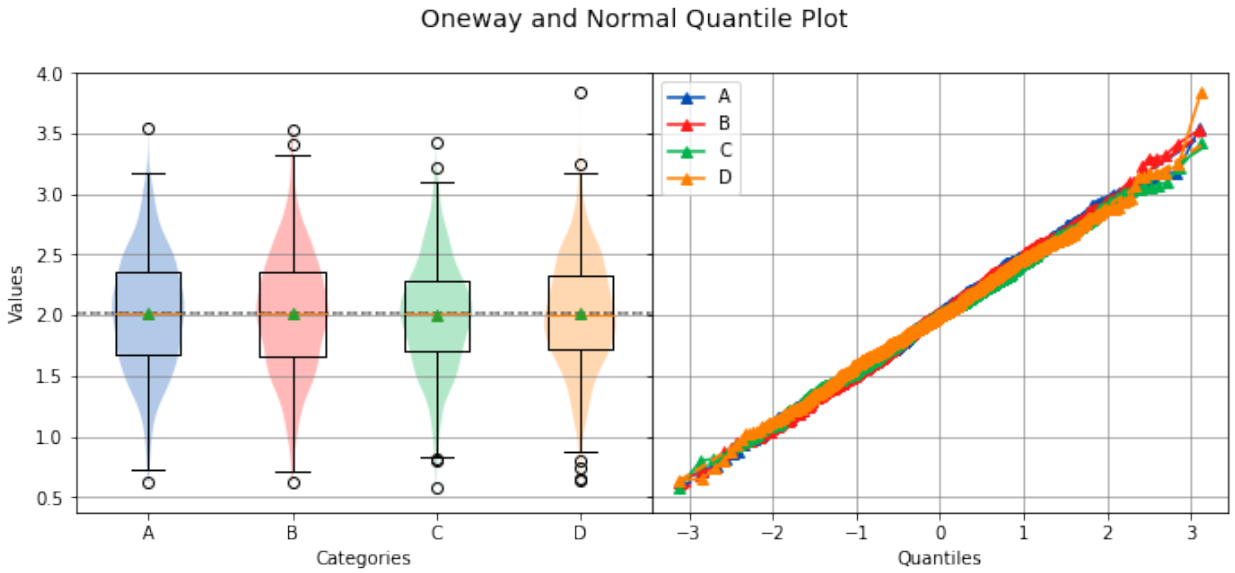
(continued from previous page)

```
T value =  5.7422
p value =  0.1248

H0: Variances are equal



Oneway ANOVA
------------

alpha   =  0.0500
f value =  0.3117
p value =  0.8169

H0: Group means are matched
```

circles

Controls whether the Tukey-Kramer circles are displayed or not. The default value is **True**.

```
analyze(
    stacked,
    groups=stacked_groups,
    circles=False,
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0118
Pooled Std Dev   =  0.4558
Grand Median     =  2.0174
```

(continues on next page)

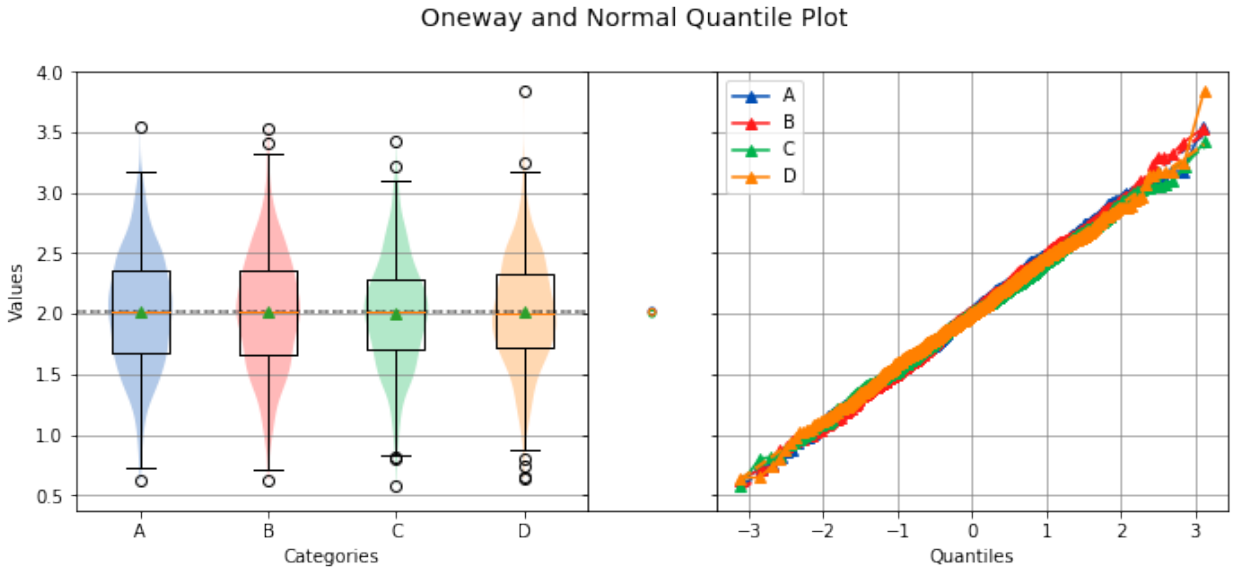
(continued from previous page)

```
Group Statistics
----------------

n              Mean           Std Dev        Min            Median         Max            ␣
↪Group
-------------------------------------------------------------------------------------------
↪------------
720            2.0211         0.4667         0.6218         2.0240         3.5506         A
736            2.0126         0.4729         0.6281         2.0228         3.5339         B
782            1.9991         0.4403         0.5786         2.0120         3.4248         C
762            2.0143         0.4439         0.6396         2.0046         3.8397         D


Bartlett Test
-------------

alpha   =  0.0500
T value =  5.7422
p value =  0.1248

H0: Variances are equal



Oneway ANOVA
------------

alpha   =  0.0500
f value =  0.3117
p value =  0.8169

H0: Group means are matched
```

alpha

Sets the significance level to use for hypothesis testing.

```
analyze(
    stacked,
    groups=stacked_groups,
    alpha=0.01,
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0118
Pooled Std Dev   =  0.4558
Grand Median     =  2.0174


Group Statistics
----------------

n              Mean          Std Dev       Min           Median        Max           ␣
↪Group
-------------------------------------------------------------------------------------
↪------------
720            2.0211        0.4667        0.6218        2.0240        3.5506        A
736            2.0126        0.4729        0.6281        2.0228        3.5339        B
782            1.9991        0.4403        0.5786        2.0120        3.4248        C
762            2.0143        0.4439        0.6396        2.0046        3.8397        D


Bartlett Test
-------------

alpha    =  0.0100
T value  =  5.7422
p value  =  0.1248

H0: Variances are equal



Oneway ANOVA
------------
```

(continues on next page)

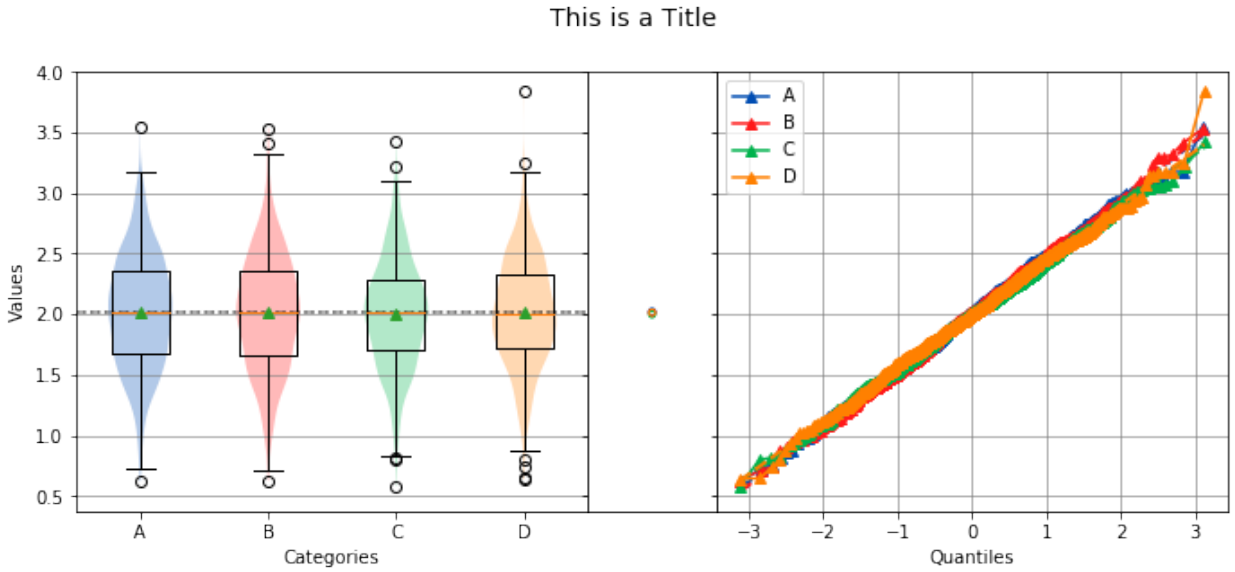
(continued from previous page)

```
alpha   =  0.0100
f value =  0.3117
p value =  0.8169

H0: Group means are matched
```

title

The title of the distribution to display above the graph.

```
analyze(
    stacked,
    groups=stacked_groups,
    title='This is a Title',
)
```




```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0118
Pooled Std Dev   =  0.4558
Grand Median     =  2.0174


Group Statistics
----------------

n              Mean           Std Dev        Min            Median         Max            ␣
↪Group
-------------------------------------------------------------------------------------------
↪------------
720            2.0211         0.4667         0.6218         2.0240         3.5506         A
```

(continues on next page)

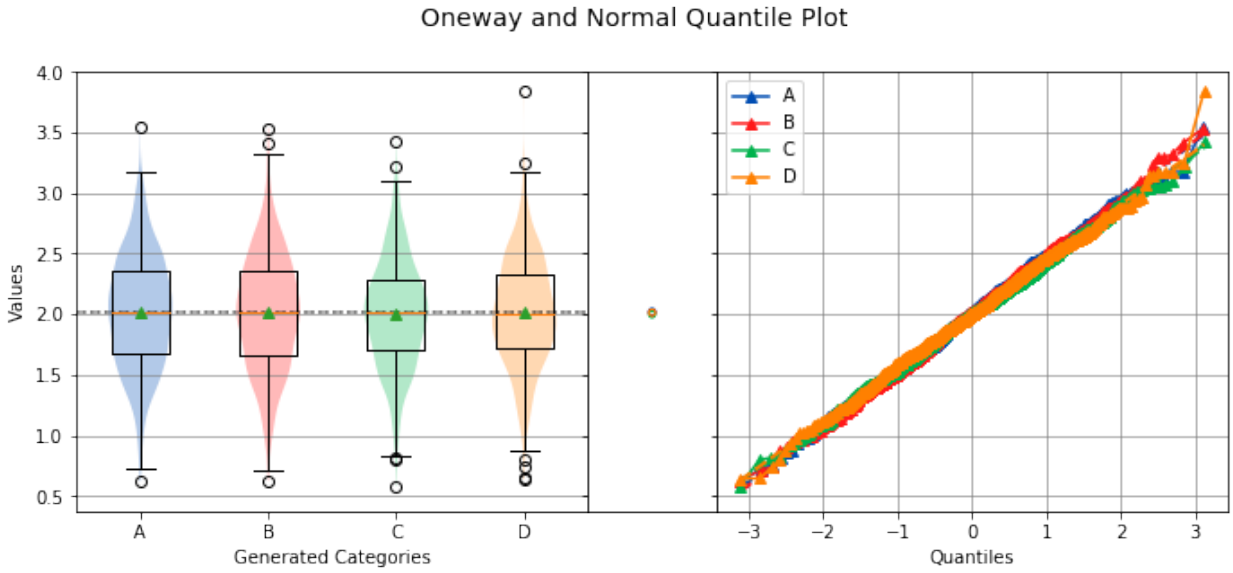
(continued from previous page)

```
736             2.0126          0.4729          0.6281          2.0228          3.5339          B
782             1.9991          0.4403          0.5786          2.0120          3.4248          C
762             2.0143          0.4439          0.6396          2.0046          3.8397          D


Bartlett Test
-------------

alpha    =  0.0500
T value  =  5.7422
p value  =  0.1248

H0: Variances are equal



Oneway ANOVA
------------

alpha    =  0.0500
f value  =  0.3117
p value  =  0.8169

H0: Group means are matched
```

categories, xname

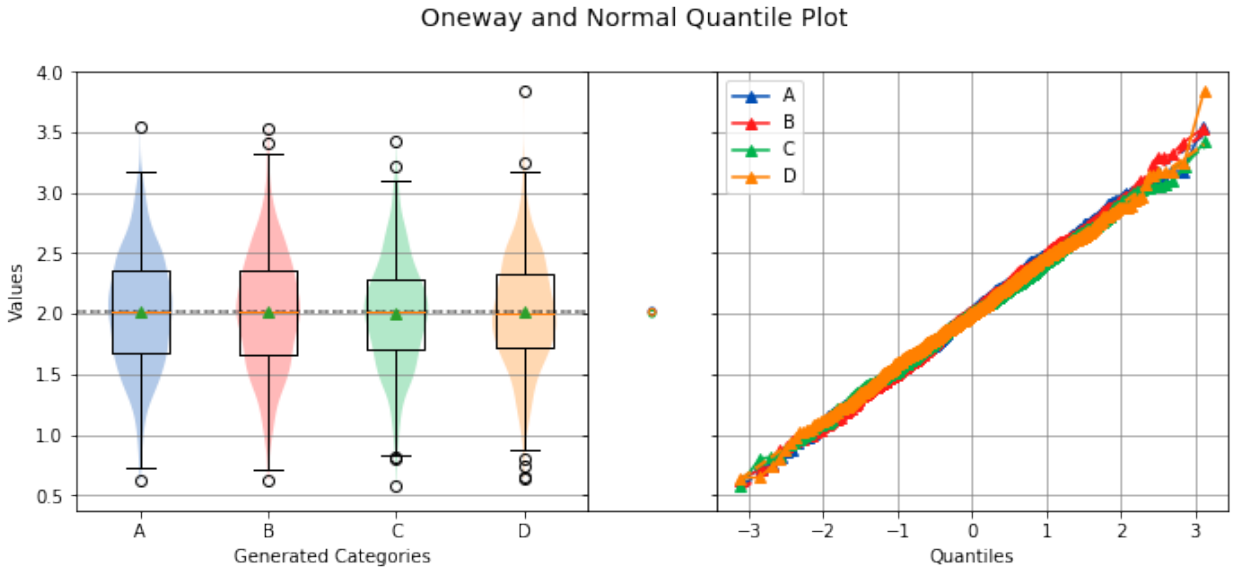
The name of the category labels to display on the x-axis.

```
analyze(
    stacked,
    groups=stacked_groups,
    categories='Generated Categories',
)
```



```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0118
Pooled Std Dev   =  0.4558
Grand Median     =  2.0174


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           Group
------------------------------------------------------------------------------------------
720           2.0211        0.4667        0.6218        2.0240        3.5506        A
736           2.0126        0.4729        0.6281        2.0228        3.5339        B
782           1.9991        0.4403        0.5786        2.0120        3.4248        C
762           2.0143        0.4439        0.6396        2.0046        3.8397        D


Bartlett Test
-------------

alpha   =  0.0500
T value =  5.7422
p value =  0.1248

H0: Variances are equal



Oneway ANOVA
------------

alpha   =  0.0500
f value =  0.3117
p value =  0.8169

H0: Group means are matched
```

```
analyze(
    stacked,
    groups=stacked_groups,
    xname='Generated Categories',
)
```



Overall Statistics

```
Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0118
Pooled Std Dev   =  0.4558
Grand Median     =  2.0174
```

Group Statistics

n	Mean	Std Dev	Min	Median	Max	Group
720	2.0211	0.4667	0.6218	2.0240	3.5506	A
736	2.0126	0.4729	0.6281	2.0228	3.5339	B
782	1.9991	0.4403	0.5786	2.0120	3.4248	C
762	2.0143	0.4439	0.6396	2.0046	3.8397	D

Bartlett Test

```
alpha   =  0.0500
T value =  5.7422
p value =  0.1248

H0: Variances are equal
```

Oneway ANOVA

(continues on next page)

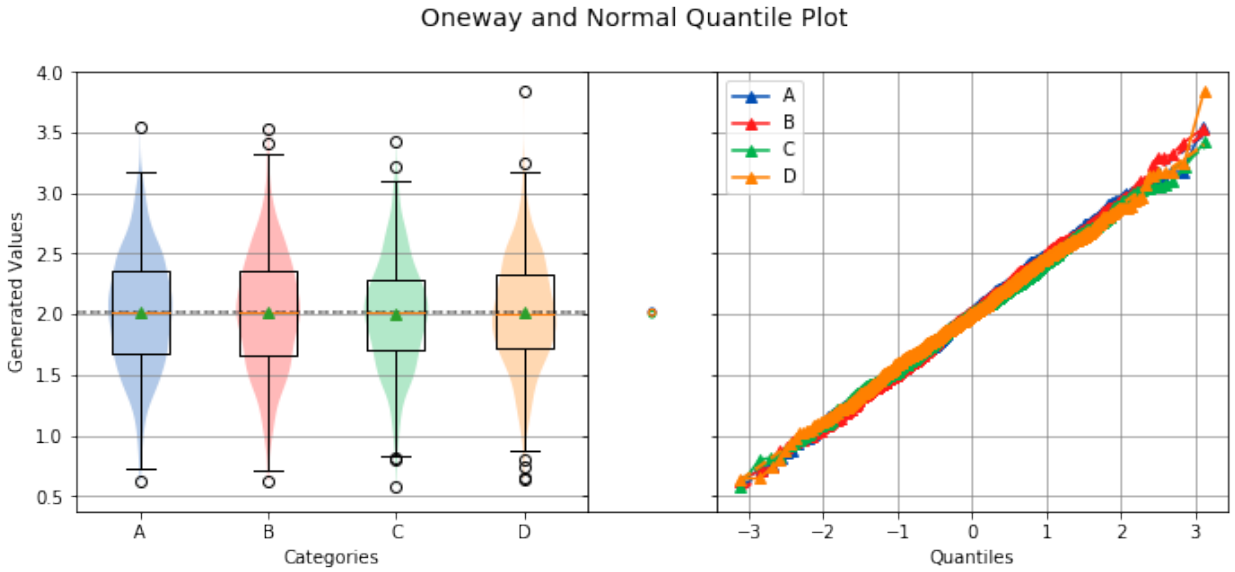
(continued from previous page)

```
alpha    =  0.0500
f value  =  0.3117
p value  =  0.8169

H0: Group means are matched
```

name, yname

The label to display on the y-axis.

```
analyze(
    stacked,
    groups=stacked_groups,
    name='Generated Values',
)
```



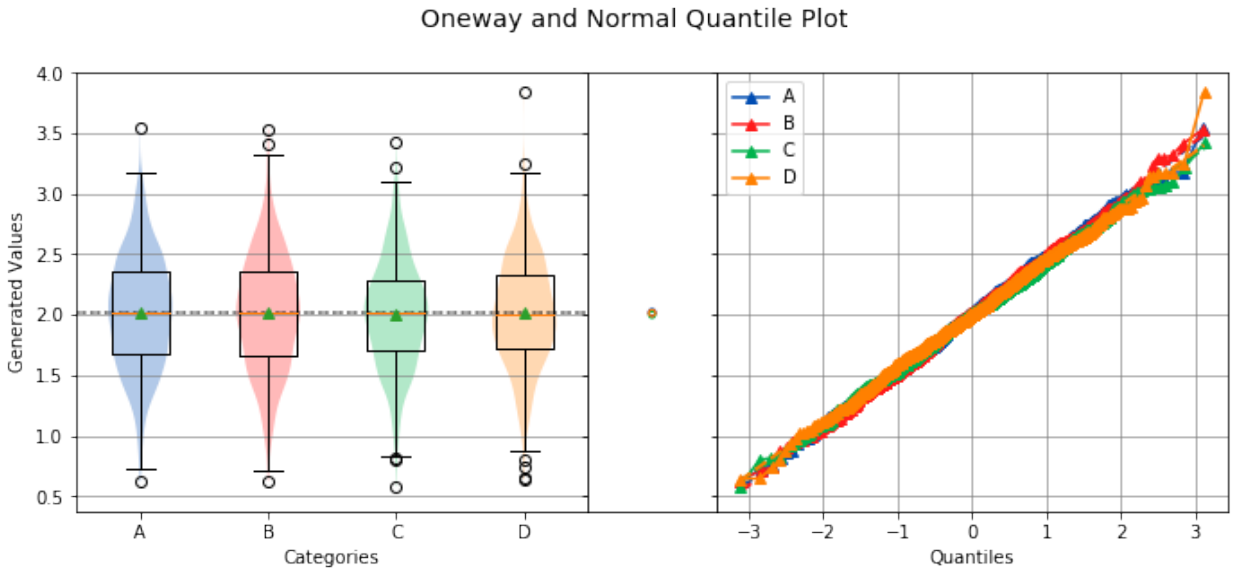

```
Overall Statistics
------------------

Number of Groups =  4
Total            =  3000
Grand Mean       =  2.0118
Pooled Std Dev   =  0.4558
Grand Median     =  2.0174


Group Statistics
----------------

n              Mean           Std Dev        Min            Median         Max
↪Group
-------------------------------------------------------------------------------------
↪------------
720            2.0211         0.4667         0.6218         2.0240         3.5506
A
```

(continues on next page)

(continued from previous page)

```
736             2.0126          0.4729          0.6281          2.0228          3.5339          B
782             1.9991          0.4403          0.5786          2.0120          3.4248          C
762             2.0143          0.4439          0.6396          2.0046          3.8397          D


Bartlett Test
-------------

alpha    =  0.0500
T value  =  5.7422
p value  =  0.1248

H0: Variances are equal



Oneway ANOVA
------------

alpha    =  0.0500
f value  =  0.3117
p value  =  0.8169

H0: Group means are matched
```

```
analyze(
    stacked,
    groups=stacked_groups,
    yname='Generated Values',
)
```




```
Overall Statistics
------------------

Number of Groups =  4
```

(continues on next page)

(continued from previous page)

```
Total            =  3000
Grand Mean       =  2.0118
Pooled Std Dev   =  0.4558
Grand Median     =  2.0174


Group Statistics
----------------

n             Mean          Std Dev       Min           Median        Max           ␣
↪Group
------------------------------------------------------------------------------------------
↪------------
720           2.0211        0.4667        0.6218        2.0240        3.5506        A
736           2.0126        0.4729        0.6281        2.0228        3.5339        B
782           1.9991        0.4403        0.5786        2.0120        3.4248        C
762           2.0143        0.4439        0.6396        2.0046        3.8397        D


Bartlett Test
-------------

alpha   =  0.0500
T value =  5.7422
p value =  0.1248

H0: Variances are equal



Oneway ANOVA
------------

alpha   =  0.0500
f value =  0.3117
p value =  0.8169

H0: Group means are matched
```

Index

A